NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

M554

DESIGN IMPLEMENTATION INTO FIELD PROGRAMMABLE GATE ARRAYS

by

Norman C. Messa

March 1991

Thesis Advisor: C. H. Lee

Approved for public release; distribution is unlimited

T253574

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO.; TASK NO; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
DESIGN IMPLEMENTATION INTO FIELD PROGRAMMABLE GATE ARRAYS

12. PERSONAL AUTHOR(S)
MESSA, Norman C.

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	1991 March	104

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Field Programmable Gate Array; Logic Cell Array

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

In the past three years a new type of programmable logic device has emerged. The programmable gate array is a new approach to an old problem of trying to implement logic designs in an efficient manner. This thesis explores the implementation of design using the Field Programmable Gate Array (FPGA). In particular, this thesis utilizes the XILINX development system tools to implement design into the XILINX Logic Cell Array (LCA). This thesis begins by defining the characteristics of the LCA and then defines the characteristics of the Small Computer Systems Interface (SCSI) which is used as a design implementation example. The XILINX implementation method is then explored and a complete design implementation study is conducted on the design example. Both Mentor Graphics and Futurenet schematic capture tools are used for design entry. Following

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL LEE, Chin-Hwa	22b. TELEPHONE (Include Area Code) 408-646-2190	22c. OFFICE SYMBOL EC/Le

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 UNCLASSIFIED

19. cont.
design implementation, backannotated design simulation is performed to study the effect of the LCA technology on design performance. The results of this thesis showed that designs implemented using this technology performed comparably to other implementation technologies. Additionally, this implementation method allows design to be completed in a significantly shorter time frame than previously possible.

Design Implementation
Into Field Programmable
Gate Arrays

by

Norman C. Messa
Lieutenant, United States Navy
B.S., Chapman College

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1991

ABSTRACT

In the past three years a new type of programmable logic device has emerged. The programmable gate array is a new approach to an old problem of trying to implement logic designs in an efficient manner. This thesis explores the implementation of design using the Field Programmable Gate Array (FPGA). In particular, this thesis utilizes the XILINX development system tools to implement design into the XILINX Logic Cell Array (LCA). This thesis begins by defining the characteristics of the LCA and then defines the characteristics of the Small Computer Systems Interface (SCSI) which is used as a design implementation example. The XILINX implementation method is then explored and a complete design implementation study is conducted on the design example. Both Mentor Graphics and Futurenet schematic capture tools are used for design entry. Following design implementation, backannotated design simulation is performed to study the effect of the LCA technology on design performance. The results of this thesis showed that designs implemented using this technology performed comparably to other implementation technologies. Additionally, this implementation method allows design to be completed in a significantly shorter time frame than previously possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to thank my Thesis Advisor Dr. Chin-Hwa Lee for his guidance and suggestions during the course of this project. I would also like to thank my second reader Dr. Murali Tummala for his many constructive suggestions. I would also like to offer a special thanks to Ms. Jennifer Tran of XILINX whose patience and perseverance were of immeasurable help in the initial stages of this project. Lastly, I would like to thank my daughter Rebecca for her Love and understanding which enabled me to complete this thesis.

I. INTRODUCTION

In this thesis the Field Programmable Gate Array (FPGA) was examined as a design medium for implementation of small to medium sized designs. The FPGA is studied here from a design implementation standpoint with emphasis concentrated on design ease, reliability and design verification. A key question of interest to be answered in this thesis is with regard to how well do designs implemented using FPGAs perform as compared to other implementation technologies. This study started by examining the FPGA technology and device characteristics. A description of the Small Computer Systems Interface (SCSI) provided the guidelines for a design implementation example discussed later in this thesis. The actual design implementation process was then examined followed by an actual design implementation example of a SCSI device which took the design from conception and schematic capture to hardware implementation. In this thesis two different design platforms were used, and a comparison of the design processes was made. Additionally, timing simulation was performed on implemented designs to examine the effects of FPGA technology on design performance.

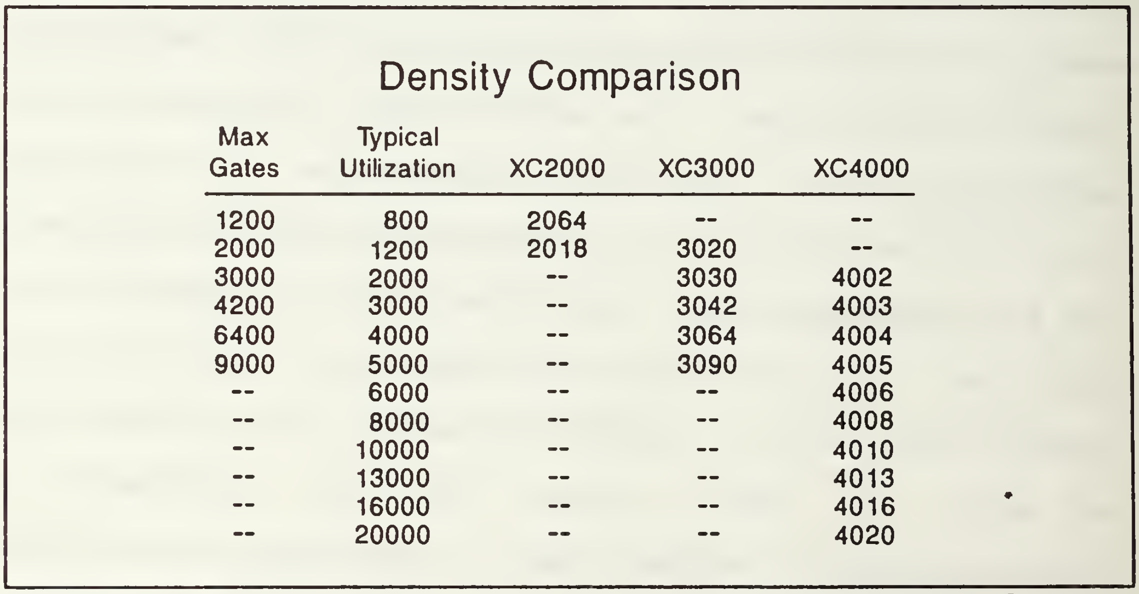
In this thesis, Chapter II will provide a description of the Logic Cell Array (LCA) technology which will be followed by a discussion of the SCSI communication protocol in Chapter

III. Chapter IV will provide insight into the actual design implementation process utilizing the XILINX development tools. Chapter V will go through a complete design implementation example from design entry to the actual design placement and routing into an LCA. Several conclusions and observations have been made as the result of this work which are summarized in Chapter VI.

II. INTRODUCTION TO FIELD PROGRAMMABLE GATE ARRAYS

A. OVERVIEW

In this chapter, the characteristics of the Field Programmable Gate Array (FPGA) is discussed with regard to its functionality, configurability and operation. Information in this chapter is derived from References 1, 2 and 3. The FPGA is a device in which small to medium sized designs may be accommodated. It is an alternative form of Application Specific Integrated Circuit (ASIC) in lieu of Programmable Logic Devices (PLDs), fixed gate arrays or full custom integrated circuits. There are currently two main technologies for FPGA. The first, XILINX, utilizes an architecture that is fully re-programmable with configuration controlled by software. The second, ACTEL, utilizes anti-fuse technology making it one time programmable. In this thesis the XILINX technology will be explored exclusively because of the re-programmability. The XILINX FPGA is refered to as a Logic Cell Array (LCA) by the industry as well as in this thesis (Logic Cell is a XILINX trademark). The XILINX LCA is an array of logic blocks that are configured and interconnected via software that is downloaded into the LCA at either power-up or any other time the designer desires in a re-program mode. This later characteristic allows the LCA's configuration to be a function of time. As the technology stands today it is currently in its second generation. The first generation XC2000 series LCAs could hold designs as large as 1800 gates whereas the second generation XC3000 series LCAs can hold designs up to 9000 gates. As to the future, the planned third generation XC4000 series LCAs will allow for designs up to 20000 gates. Both the XC2000 and XC3000 series LCAs are currently available in production quantities. The XC4000 series LCAs will be available sometime within the next year. Figure 1 shows the density comparison of the three generations of XILINX LCA's. In this thesis design consideration was given to both XC2000 and XC3000 LCA's.



Density Comparison

Max Gates	Typical Utilization	XC2000	XC3000	XC4000
1200	800	2064	--	--
2000	1200	2018	3020	--
3000	2000	--	3030	4002
4200	3000	--	3042	4003
6400	4000	--	3064	4004
9000	5000	--	3090	4005
--	6000	--	--	4006
--	8000	--	--	4008
--	10000	--	--	4010
--	13000	--	--	4013
--	16000	--	--	4016
--	20000	--	--	4020

Figure 1 Density comparison for three generations of LCA's [from Ref. 3]

B. ARCHITECTURE OF THE LOGIC CELL ARRAY (LCA)

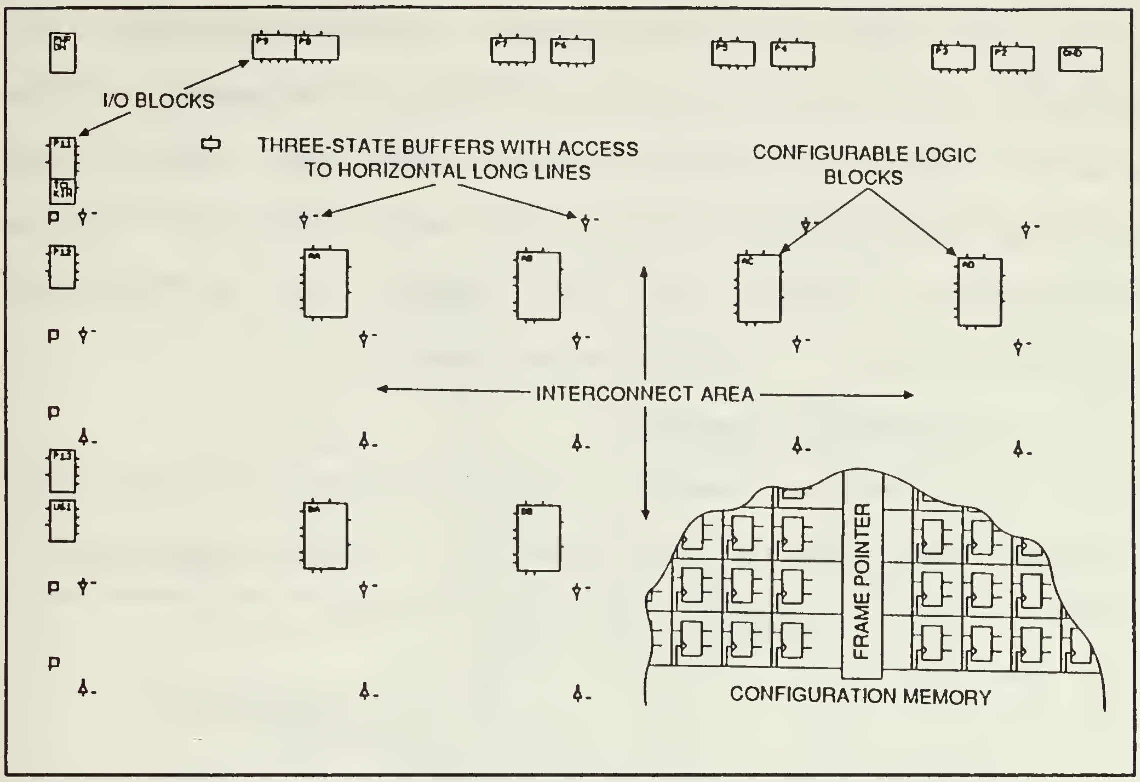

I/O BLOCKS
THREE-STATE BUFFERS WITH ACCESS TO HORIZONTAL LONG LINES
CONFIGURABLE LOGIC BLOCKS
INTERCONNECT AREA
FRAME POINTER
CONFIGURATION MEMORY

Figure 2 General structure of XC3020 LCA [from Ref. 1]

The XILINX LCA architecture consists of three major components:

- Configurable Logic Blocks (CLBs)
- Input/Output Blocks (IOBs)
- Interconnect

In addition, configuration memory is used to hold the configuration program bits which control the configuration of CLBs, IOBs and interconnect. Figure 2 shows the general structure of the XC3000 family LCA. On the perimeter, configurable Input/Output Blocks (IOBs) provide the interface between the package pins and the internal array of configurable logic. The Configurable Logic Blocks (CLBs) are arranged in an array with the interconnect programmed to form networks carrying signals between IOBs and CLBs. The functions implemented in the LCA are controlled by a configuration program which is loaded into an internal distributed array of configuration memory.

1. Configuration memory

Configuration memory consists of a distributed array of static memory cells (see Figure 3). During configuration

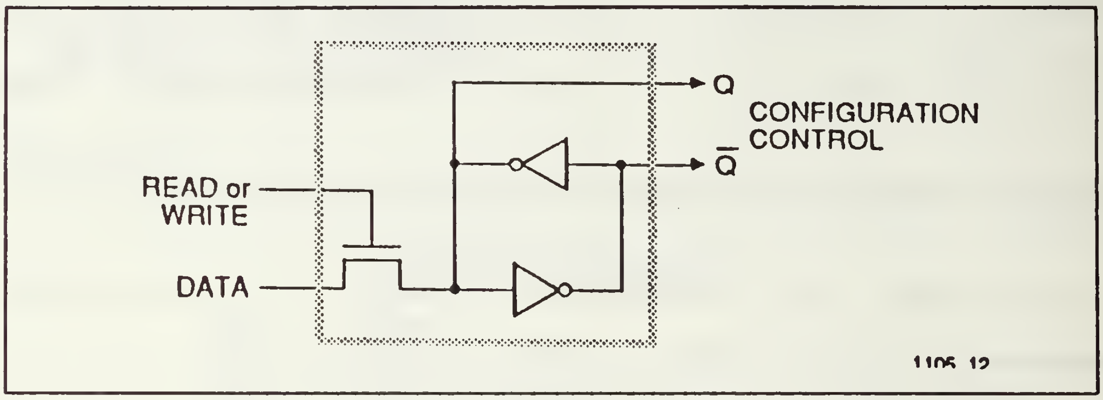


Figure 3 Configuration static memory cell [from Ref. 1]

the cell is written through the data line and is read through the data line during readback. During normal operation the pass transistor is off, and continuous configuration control is provided. There are five methods for loading configuration program data into configuration memory. Two methods load the data serially and three methods load the data in a byte wide parallel manner.

2. Input/output blocks (IOBs)

Figure 4 shows an IOB for an XC3000 family LCA. It provides the interface between the external package pin and the internal configurable logic. It is also the means by which the configuration program is loaded into the LCA during the program or re-program phases of LCA operation. It allows

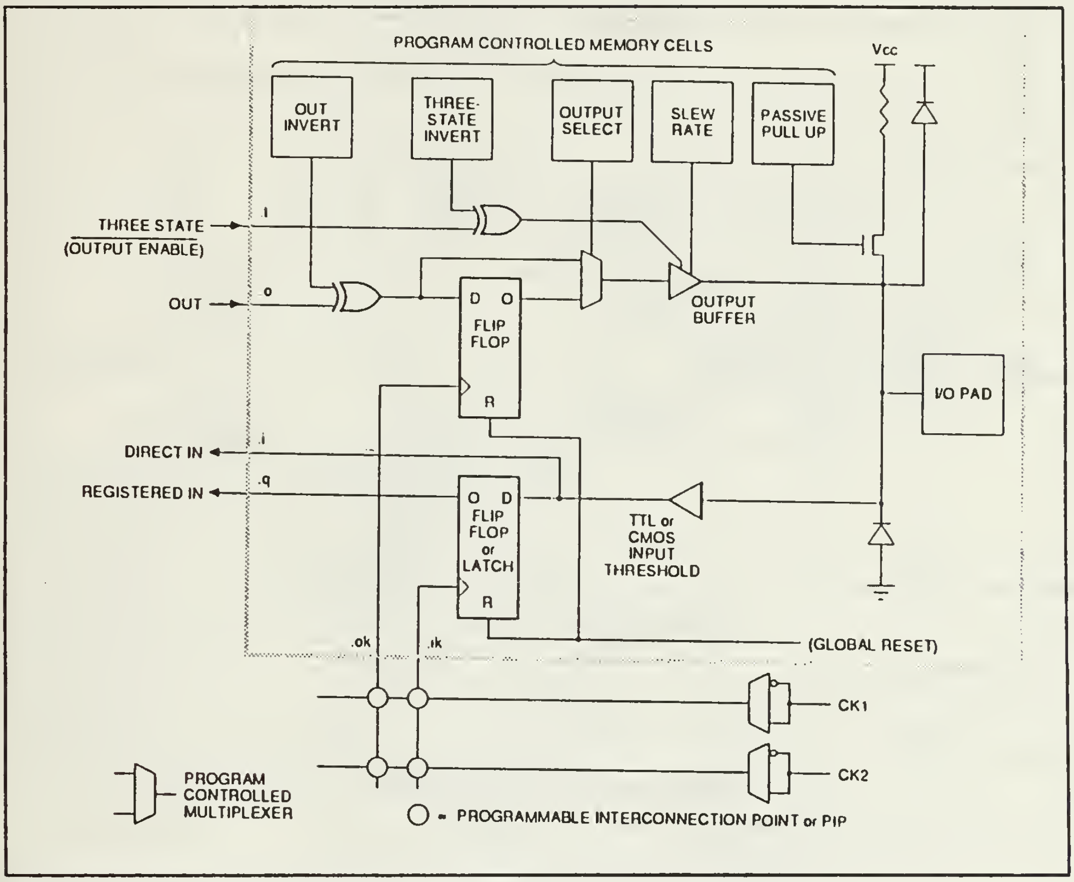


Figure 4 XC3000 series LCA IOB [from Ref.1]

for either registered or direct inputs. Each IOB has a programmable tri-state output buffer that can be driven by either a registered or a direct output signal. Specific configuration of the IOB is determined by the contents of the program controlled memory cells. A global reset is also incorporated. This global reset is important on the initial configuration to have all flip-flops reset for proper and reliable operation.

3. Configurable logic blocks (CLBs)

Figure 5 shows a Configurable Logic Block (CLB) for an XC3000 family LCA. The CLB's provide the functional units

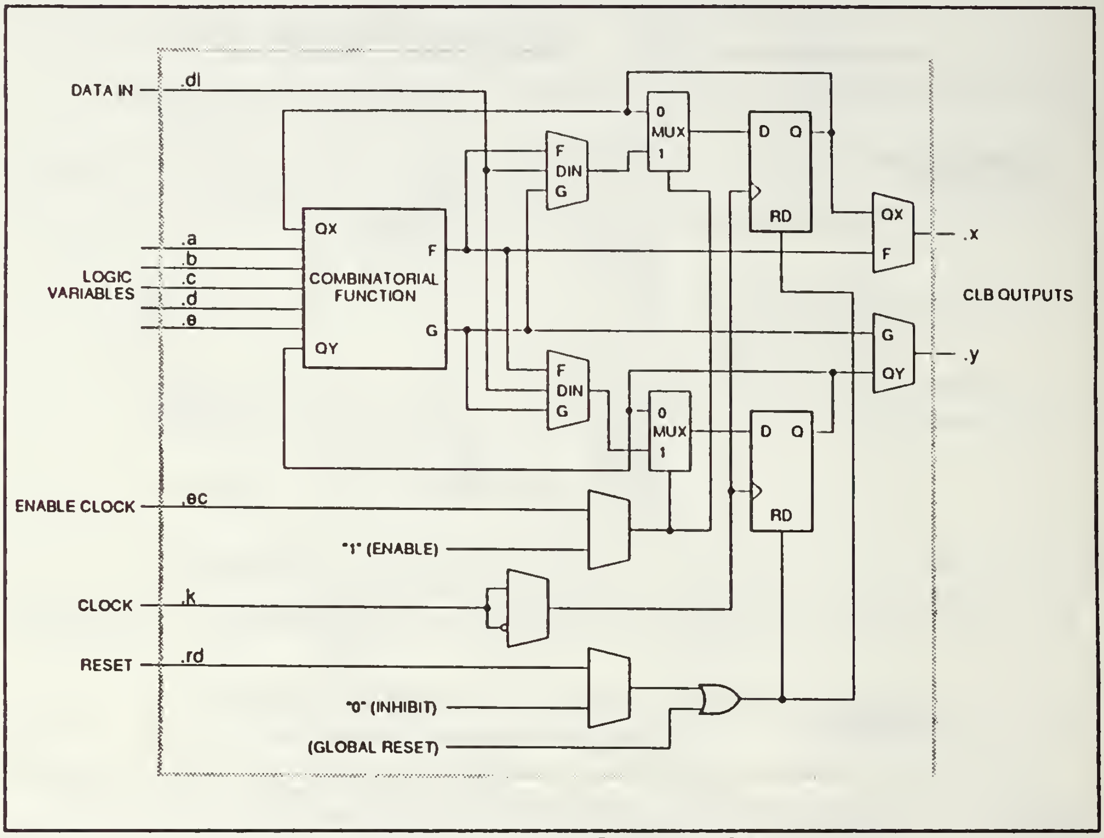


Figure 5 XC3000 family LCA Configurable Logic Block [from Ref. 1]

through which the user's design is implemented. The CLBs are arranged in an array (8 x 8 in the case of the XC3020) with two letter designations corresponding to the row and column where they are situated. The XC3000 family LCAs accomodate up to five logic variables. The combinatorial logic section of the CLB utilizes a 32 by 1 lookup table to implement boolean functions. Figure 6 shows the combinational logic types available.

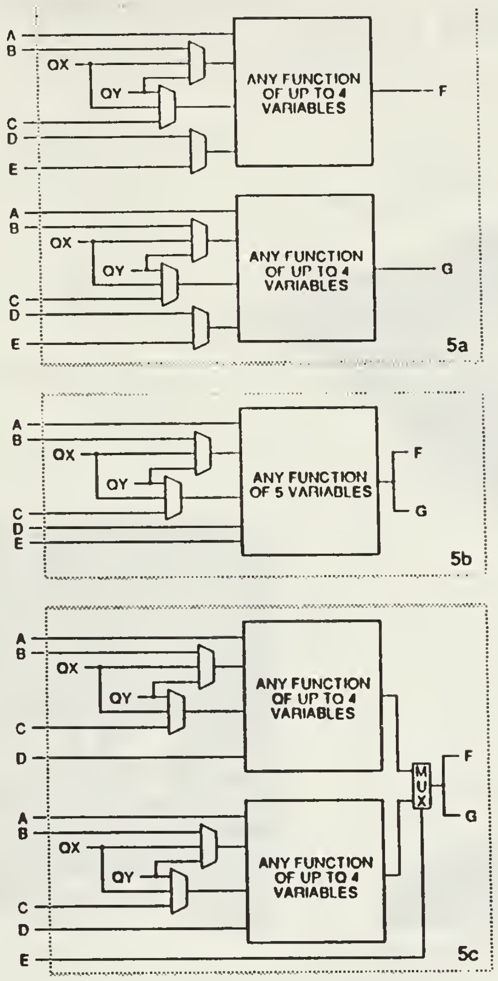

A
B
QX
QY
C
D
E
ANY FUNCTION OF UP TO 4 VARIABLES
F
G
5a
ANY FUNCTION OF 5 VARIABLES
5b
MUX
5c

Figure 6 Combinational logic types available in XC3000 family LCA [from Ref. 1]

4. Interconnect

The programmable interconnects in the LCA serve to connect inputs, outputs and CLBs into logical networks. CLB interconnects are physically comprised of two-layer metalization. In Figure 7 the pass-transistors, each controlled by a configuration bit from the configuration program form

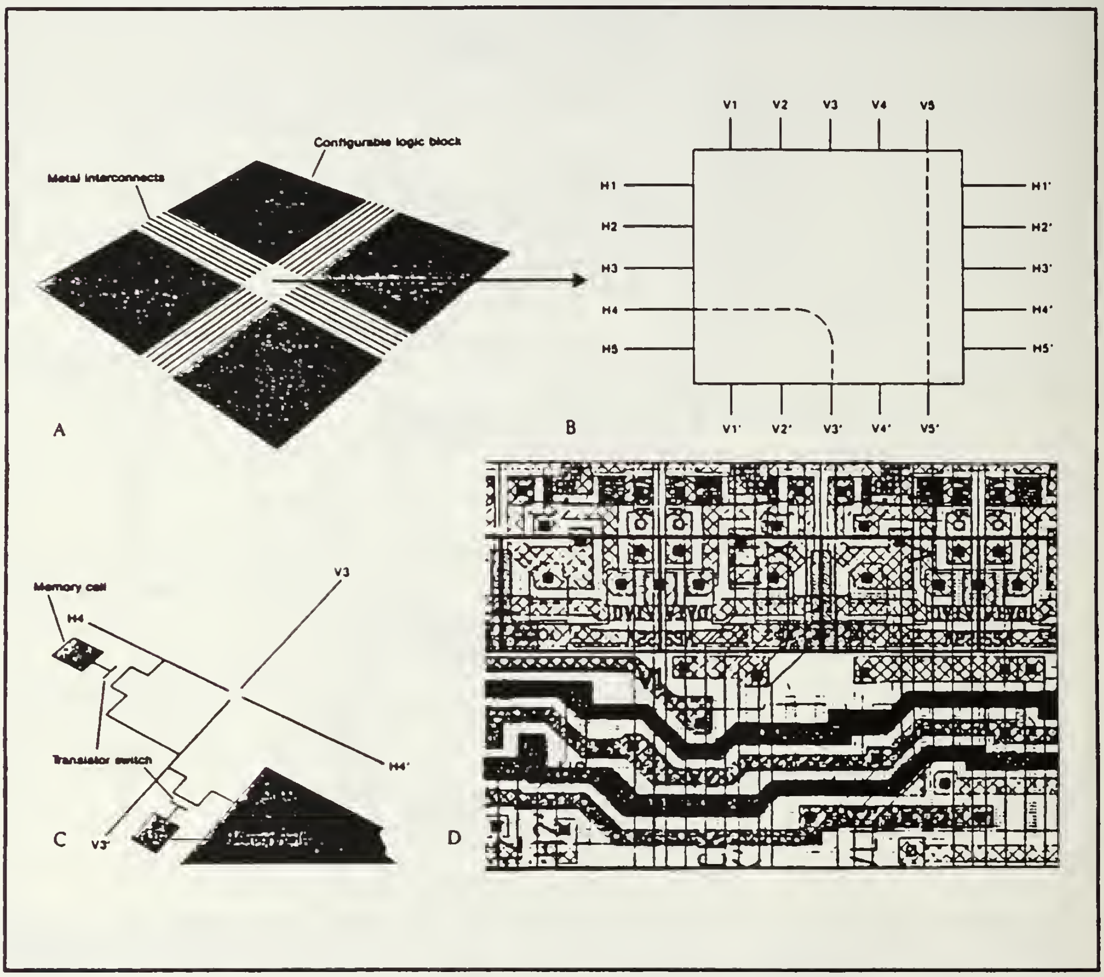


Figure 7 LCA interconnect and switching matrix [from Ref. 3]

Programmable Interconnect Points (PIPs) and switching matrices make-up the connections between the metal segments and the CLB pins (see Figure 8). There are three types of programmable interconnect available in the LCA:

- General Purpose Interconnect
- Direct Interconnect
- Long Lines

The general purpose interconnects shown in Figure 8 are metal

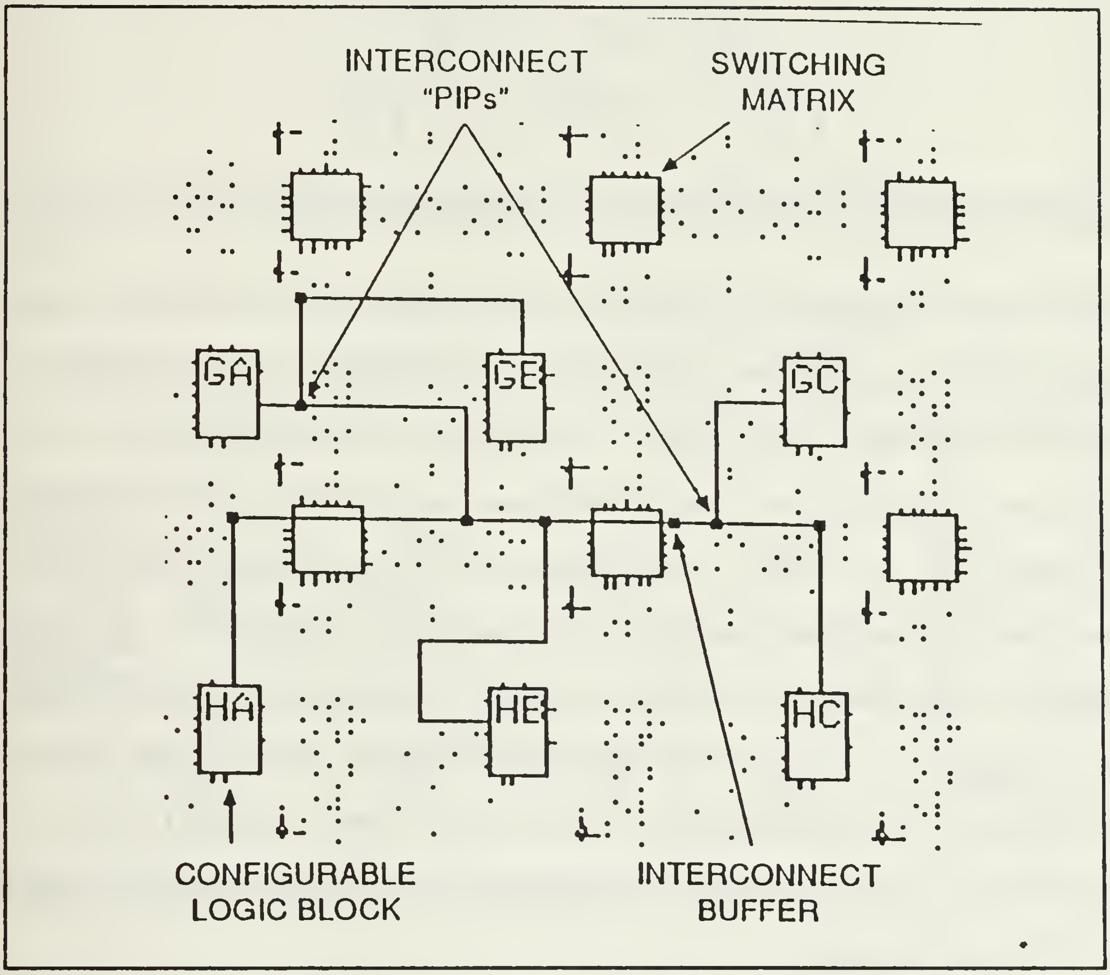


Figure 8 LCA PIP's and switching matrix [from Ref. 1]

segments that run between the rows and columns of CLBs joined on each end by a switching matrix shown in Figures 9 and 10. Switching matrix connections are controlled by bits in

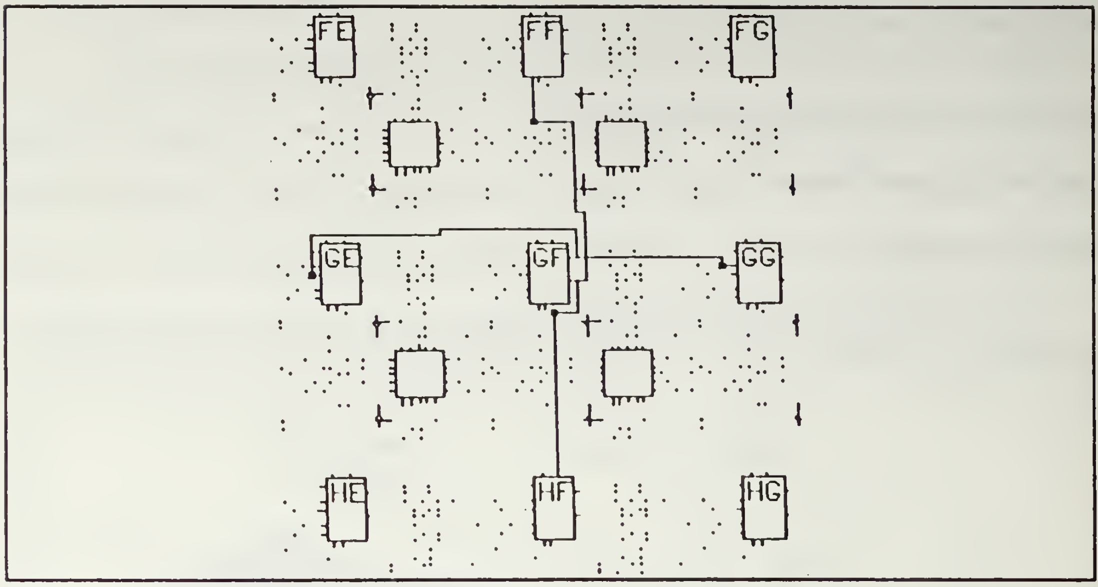


Figure 9 Direct interconnects [from Ref. 1]

configuration memory. Direct interconnects are high speed segments for connecting IOBs with nearby CLBs (see Figure 9). Direct interconnect can also be used to connect adjacent CLBs. Long lines are special lines that run nearly the entire length and width of the chip and bypass all switching matrices. These can be used to transfer signals that must be routed a long distance. This is done to minimize signal skew (see Figure 11). Figure 12 shows the entire XC3020 LCA which is capable of implementing designs up to 2000 gates in size. Additionally, the LCA has the capability of providing its own internal oscillator.

C. MODES OF CONFIGURATION

There are five modes of operation that the LCA may be configured. These modes specifically deal with how the

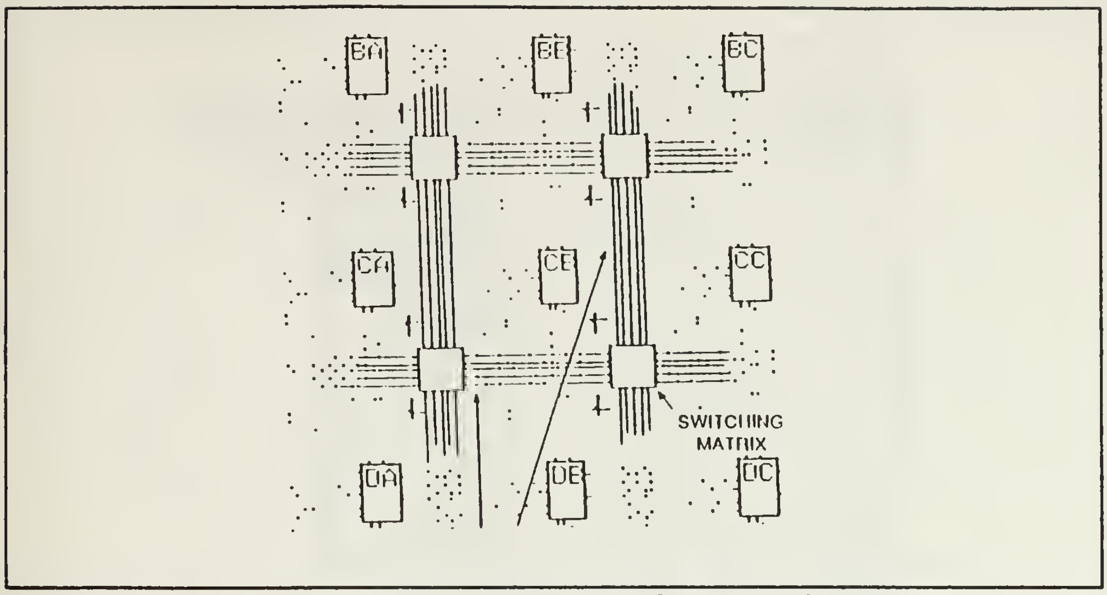


Figure 10 General interconnect and switching matrix [from Ref. 1]

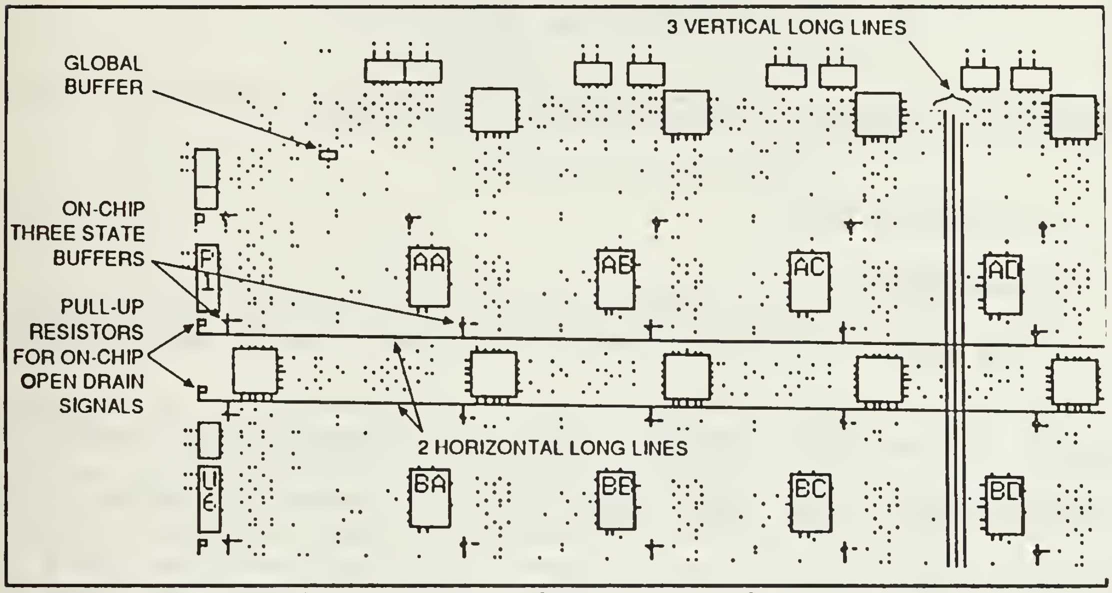
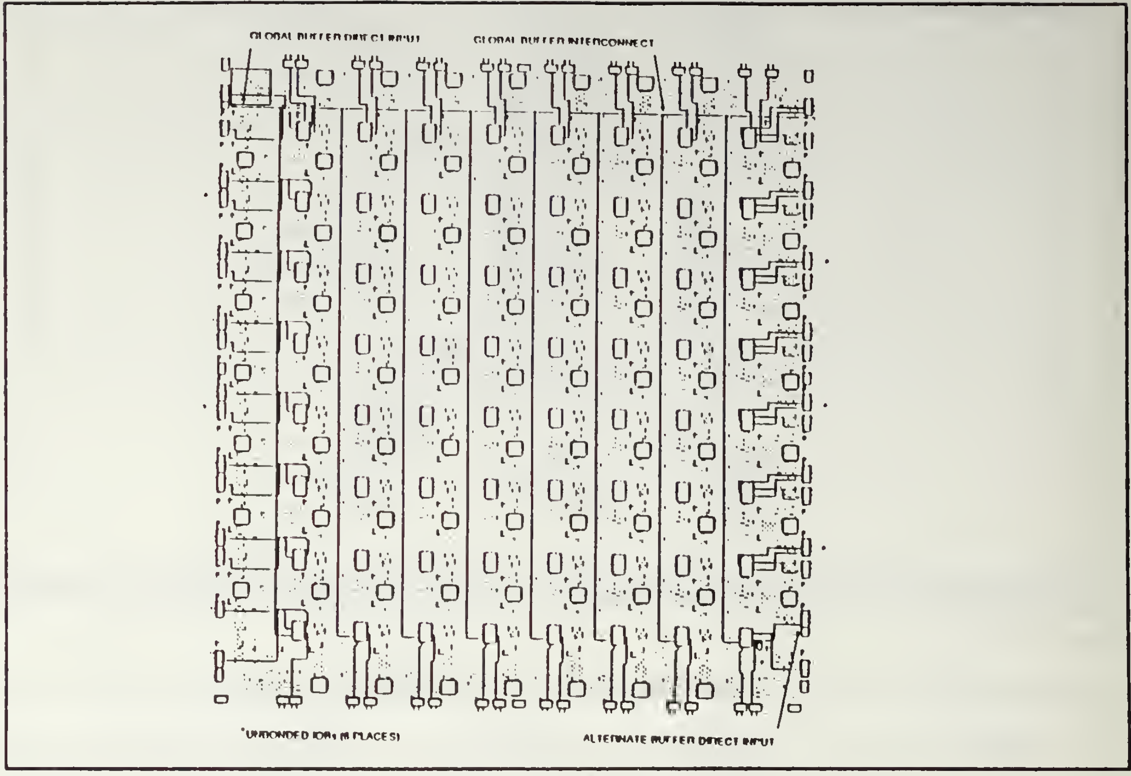

3 VERTICAL LONG LINES
GLOBAL BUFFER
ON-CHIP THREE STATE BUFFERS
PULL-UP RESISTORS FOR ON-CHIP OPEN DRAIN SIGNALS
2 HORIZONTAL LONG LINES
AA
AB
AC
AD
BA
BB
BC
BD

Figure 11 LCA long lines [from Ref. 1]

configuration program is transferred from some external device into the LCA's configuration memory. The five modes are:

- Master Serial Mode
- Master Parallel Mode


GLOBAL BUFFER DIRECT INPUT
GLOBAL BUFFER INTERCONNECT
*UNBONDED IOBs (8 PLACES)
ALTERNATE BUFFER DIRECT INPUT
Figure 12 XC3020 LCA structure [from Ref. 1]

- Peripheral Mode
- Slave Mode
- Daisy Chain Mode (with Master or Peripheral Mode leader)

In any of the Master modes, the LCA automatically loads its configuration data from some external memory device. In Master Serial Mode (see Figure 13) configuration data is loaded into the LCA via the DATA IN pin (DIN) from a synchronous serial source. XILINX can provide a serial configuration PROM specifically for that purpose. In Master Parallel Mode (see Figure 14) configuration data is provided a byte at a time to the D0-D7 pins as a result of a 16 bit address generated by the LCA from pins A0-A15. There are two

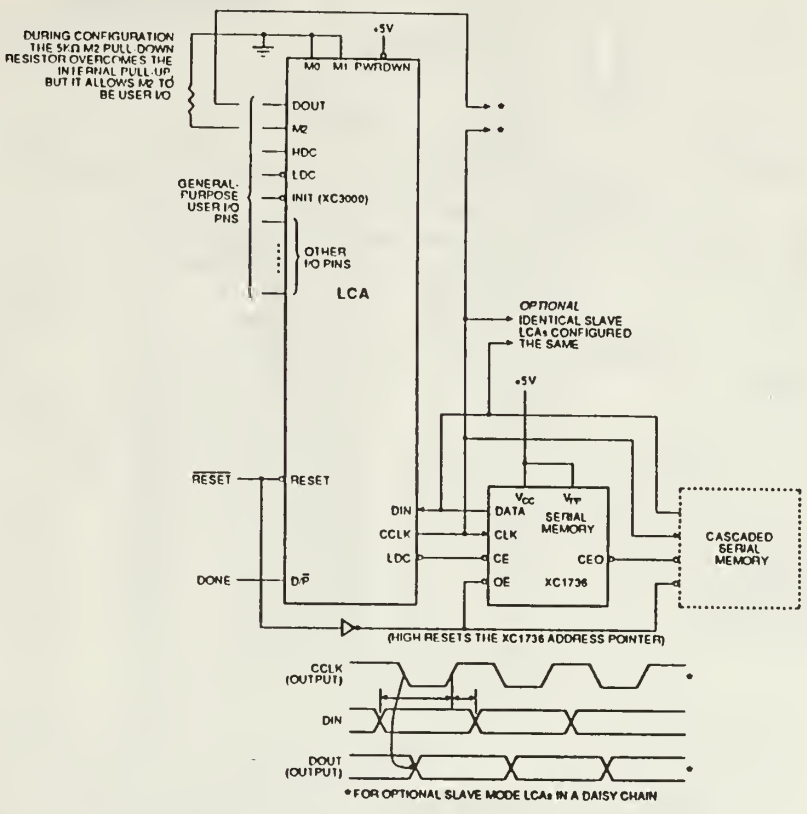

DURING CONFIGURATION THE 5KΩ M2 PULL-DOWN RESISTOR OVERCOMES THE INTERNAL PULL-UP, BUT IT ALLOWS M2 TO BE USER I/O
+5V
M0 M1 PWRDWN
DOUT
M2
HDC
LDC
INIT (XC3000)
GENERAL-PURPOSE USER I/O PINS
OTHER I/O PINS
LCA
OPTIONAL IDENTICAL SLAVE LCAs CONFIGURED THE SAME
+5V
RESET
RESET
DIN
CCLK
LDC
DONE
D/P
VCC VPP
DATA
SERIAL MEMORY
CLK
CE
CEO
OE
XC1736
CASCADED SERIAL MEMORY
(HIGH RESETS THE XC1736 ADDRESS POINTER)
CCLK (OUTPUT)
DIN
DOUT (OUTPUT)
* FOR OPTIONAL SLAVE MODE LCAs IN A DAISY CHAIN

Figure 13 Master Serial configured LCA [from Ref. 1]

variations to the Master Parallel mode. Master Parallel Low brings configuration data into the LCA starting at address 0000 while Master Parallel High brings configuration data into the LCA starting at address FFFF. This provides compatibility with microprocessors that begin program execution from low memory and increment as well as those that begin program execution from high memory and decrement. The final Master Mode variation is Master Mode (Serial or Parallel) with daisy-chained slaves (see Figure 15). The configuration clock

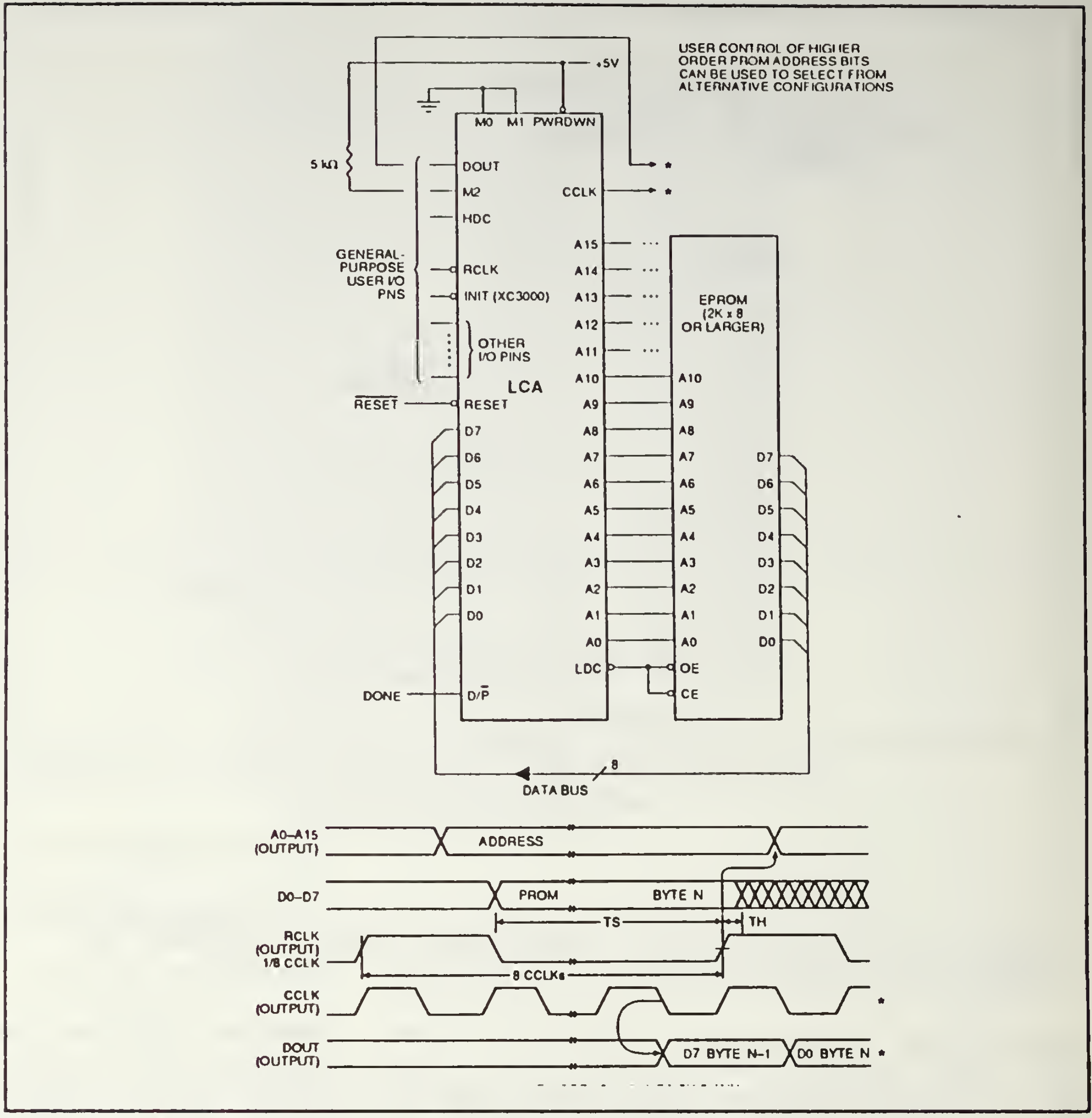


Figure 14 Master Parallel Mode configured LCA [from Ref. 1]

(CCLK) from the master LCA is provided to the slave LCAs and their serialized data is passed from DOUT to DIN down the daisy-chain. The serial configuration bit stream is passed down the daisy-chain synchronized to CCLK. On the leading edge of CCLK data comes into DIN and is passed to DOUT on the trailing edge of CCLK. In Peripheral Mode (see Figure 16) the

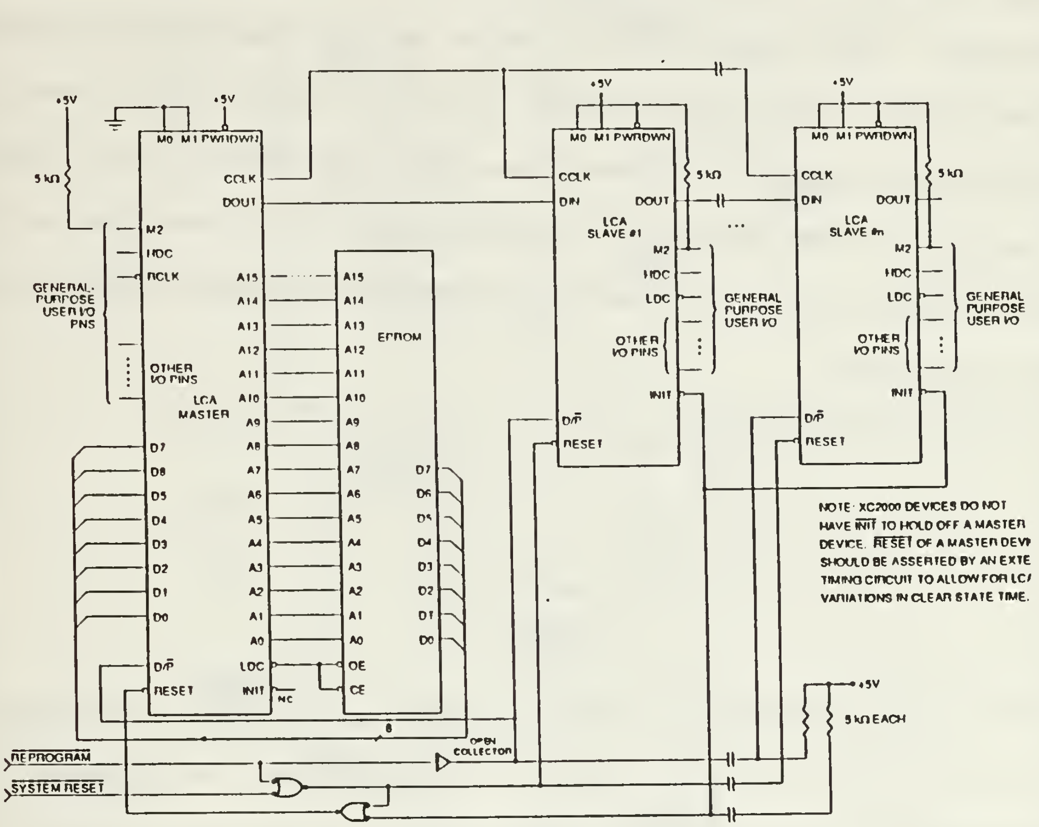


Figure 15 Daisy-chained Slave Mode LCA's [from Ref.1]

LCA is treated as a processor peripheral and configuration data is sent to it in a byte-wide fashion. A byte of configuration data is absorbed by the LCA for each processor write cycle. In this mode, daisy-chaining with slave LCAs is also permitted. This mode is most useful since it allows the LCA to be reconfigured at any time. The implication of this is that an LCA can be used to serve many functions and occupy the same space. This would be important in any space limited situation. The final mode in which an LCA can be configured is the Slave Mode (see Figure 17). In this mode, serial configuration data is strobed into the LCA one bit at a time. The source of the configuration data may be a previous slave LCA in a daisy-chain, a Master or Peripheral Mode LCA, or any other processor.

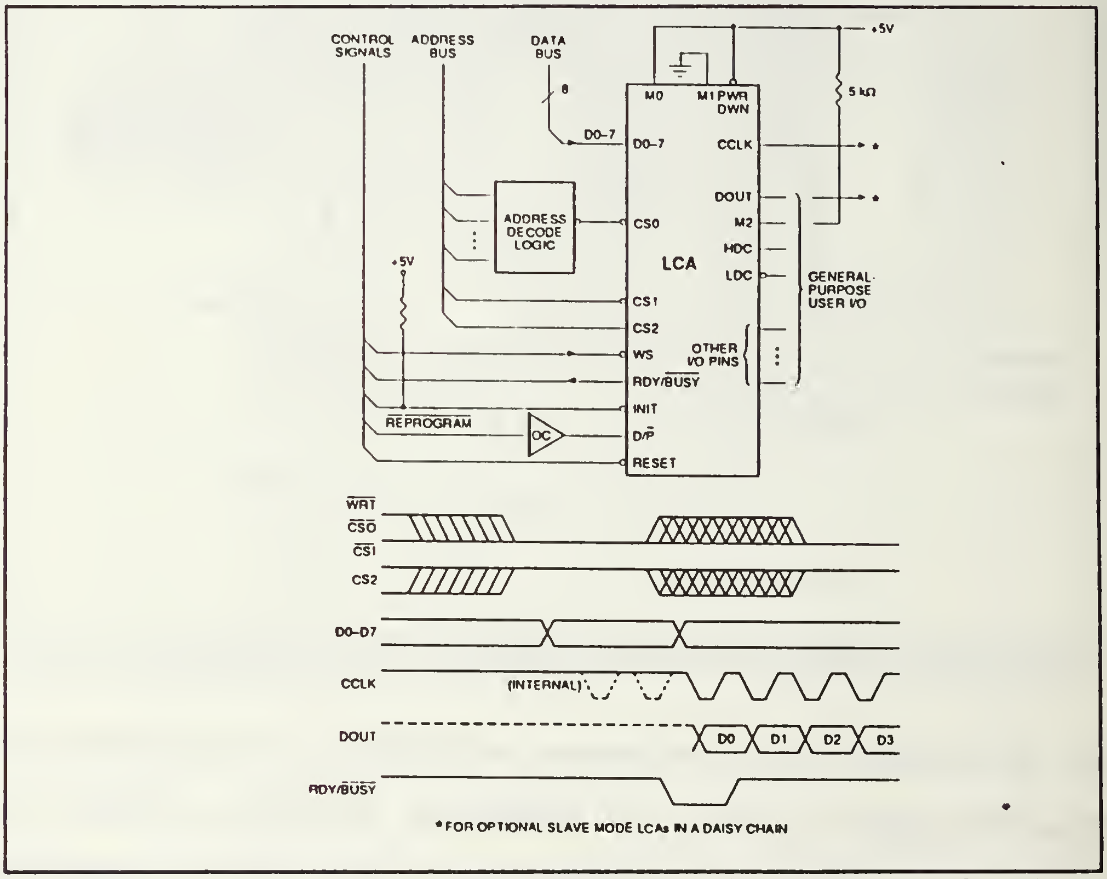


Figure 16 Peripheral Mode configuration [from Ref. 1]

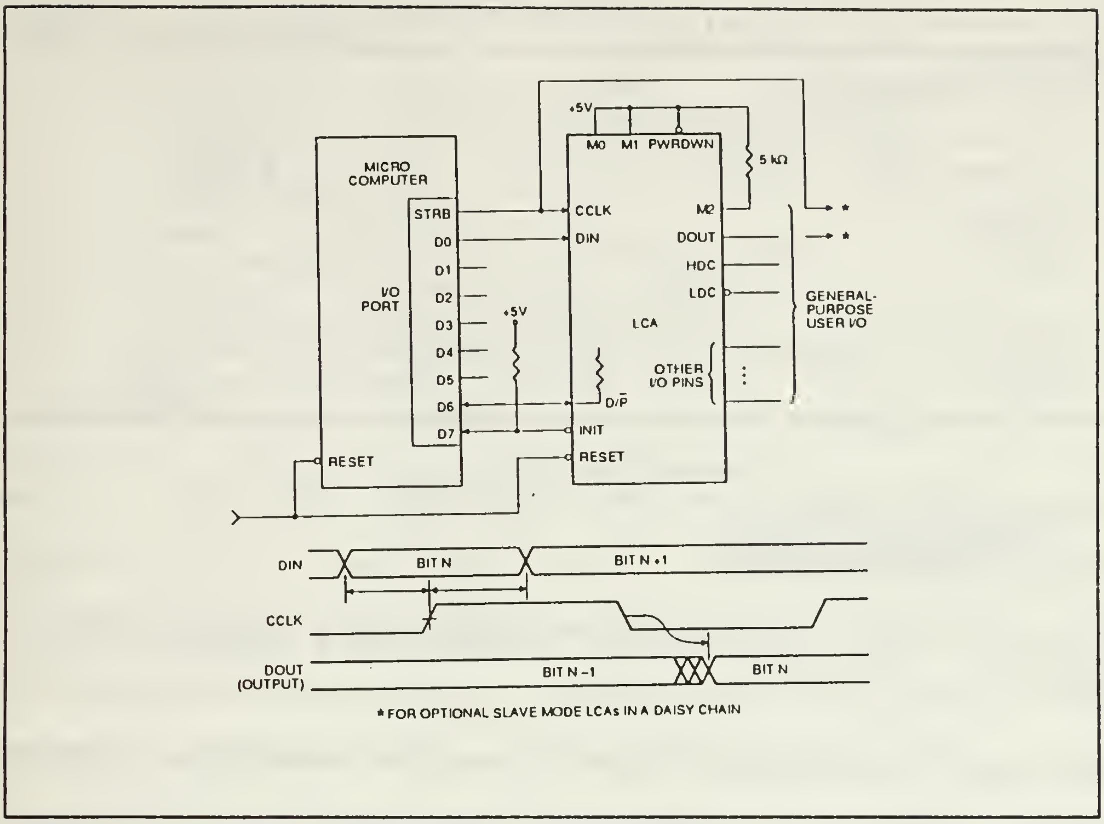


Figure 17 Slave Mode configuration [from Ref. 1]

D. PROGRAMMING THE LCA

Figure 18 shows a state diagram for the LCA configuration process. The mode with which the LCA is configured is determined by the input level on the three mode pins; M0, M1 and M2. Table 1 summarizes the configuration alternatives. It should be noted that configuration of an LCA is one trait that separates it from the more conventional logic devices (i.e., PLDs and full custom Gate Arrays). The LCA is configured rather than programmed although the two terms have been used interchangeably. The programming is done externally to the LCA in either a fixed memory device like an Erasable Programmable Read Only Memory (EPROM), a serial PROM, or a bit

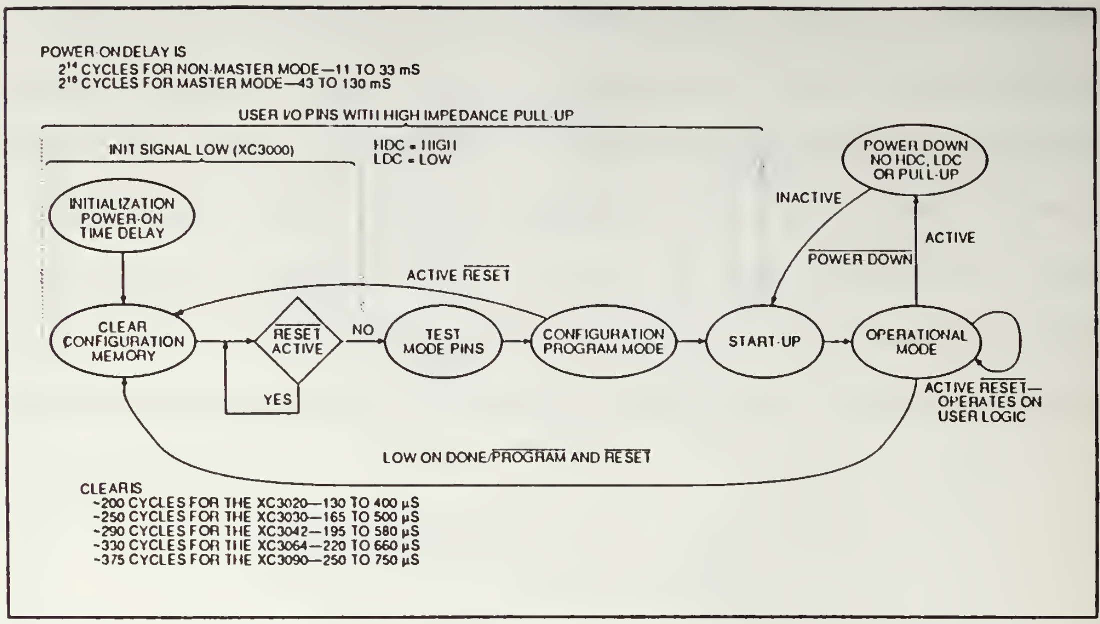

POWER-ON DELAY IS
2^{14} CYCLES FOR NON-MASTER MODE—11 TO 33 mS
2^{16} CYCLES FOR MASTER MODE—43 TO 130 mS
USER I/O PINS WITH HIGH IMPEDANCE PULL-UP
INIT SIGNAL LOW (XC3000)
HDC = HIGH
LDC = LOW
INITIALIZATION POWER-ON TIME DELAY
POWER DOWN NO HDC, LDC OR PULL-UP
INACTIVE
ACTIVE
$\overline{\text{POWER DOWN}}$
ACTIVE $\overline{\text{RESET}}$
CLEAR CONFIGURATION MEMORY
$\overline{\text{RESET}}$ ACTIVE
NO
YES
TEST MODE PINS
CONFIGURATION PROGRAM MODE
START-UP
OPERATIONAL MODE
ACTIVE $\overline{\text{RESET}}$—OPERATES ON USER LOGIC
LOW ON DONE/$\overline{\text{PROGRAM}}$ AND $\overline{\text{RESET}}$
CLEAR IS
~200 CYCLES FOR THE XC3020—130 TO 400 µS
~250 CYCLES FOR THE XC3030—165 TO 500 µS
~290 CYCLES FOR THE XC3042—195 TO 580 µS
~330 CYCLES FOR THE XC3064—220 TO 660 µS
~375 CYCLES FOR THE XC3090—250 TO 750 µS

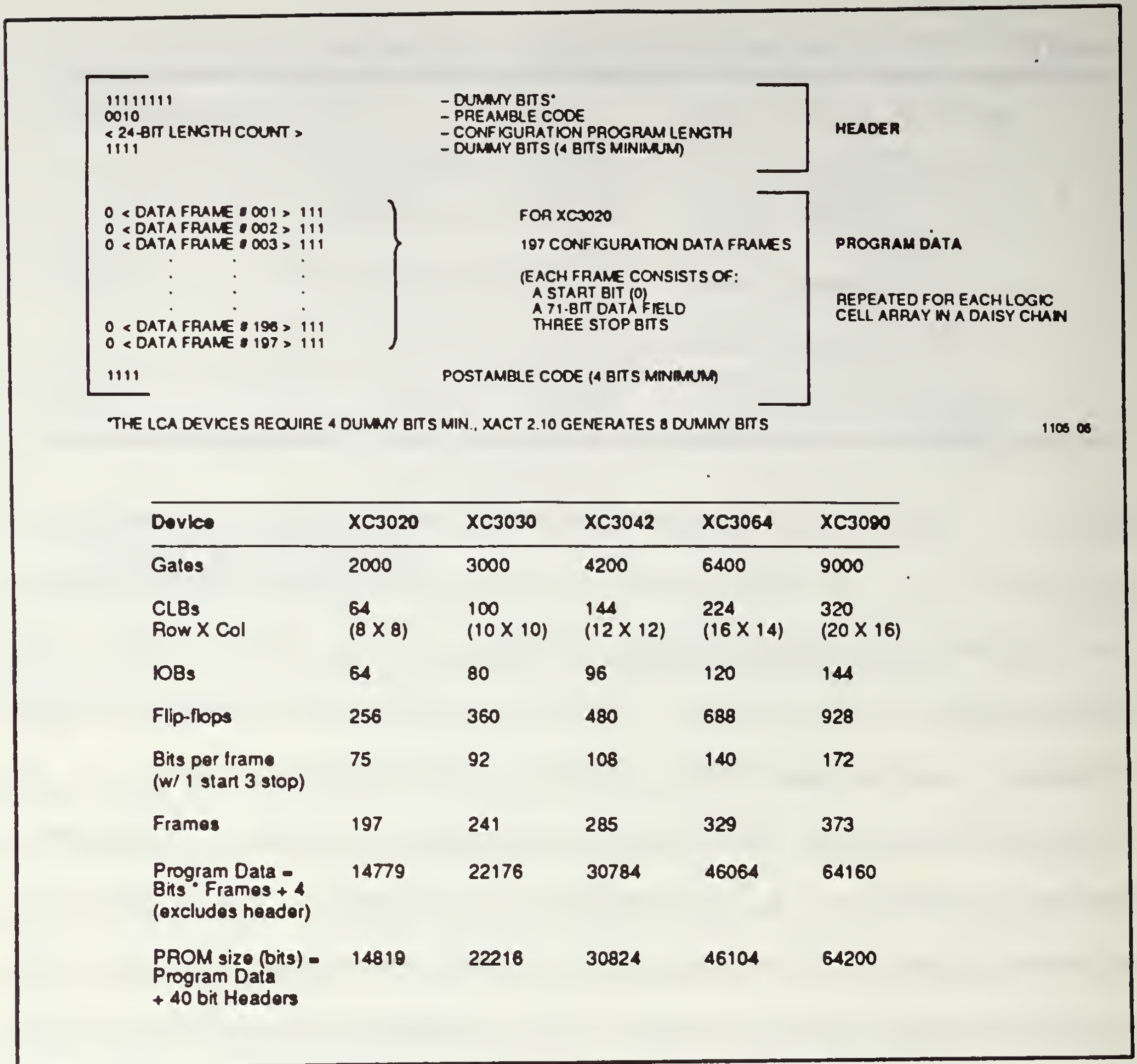
Figure 18 LCA configuration process state diagram [from Ref. 1]

stream that is stored in either memory or some other type of storage medium. Configuration of the LCA involves trans-transferring the configuration bit stream into the LCA's configuration memory cells. The first step in programming the LCA is the INITIALIZATION phase. The initialization state will continue until a 14-bit timer running off of a 1 MHz clock has timed out. This timer clock is assumed to have a variation of $\pm 50\%$ due to differences in process and temperature which will allow the timer to count at a rate of 0.5 MHz to 1.5 MHz. This equates to an INITIALIZATION phase of 11 to 33 ns. In the Master Modes this time is extended by a factor of four to ensure that any slave devices the LCA may be driving are initialized. At the end of initialization the LCA enters the CLEAR phase where it clears configuration

Table I LCA CONFIGURATION MODES [from Ref. 1]

M0	M1	M2	Clock	Mode	Data
0	0	0	active	Master	Bit Serial
0	0	1	active	Master	Byte Wide Addr. = 0000 up
0	1	0	—	reserved	—
0	1	1	active	Master	Byte Wide Addr. = FFFF down
1	0	0	—	reserved	—
1	0	1	passive	Peripheral	Byte Wide
1	1	0	—	reserved	—
1	1	1	passive	Slave	Bit Serial

memory. When the INITIALIZATION and CLEAR phases have completed, it is indicated by an active low INIT signal (available only on XC3000 family). At this point the CONFIGURATION phase begins. The configuration program header contains a length count of the configuration data to be transferred to the configuration memory. Figure 19 shows a typical format of a configuration program. When LCAs are daisy-chained, the preamble and length count are shifted into the LCA on the leading edge of the configuration clock (CCLK) and shifted out on trailing clock edges. Once an LCA has received a preamble and length count, DOUT goes high until the LCA has absorbed the appropriate number of frames. This method allows several configuration programs to be stored on one EPROM. When configuration memory is full and the length count agrees, the LCA will startup and become operational. During startup user I/O pins become active and can be defined to be either TTL or CMOS compatible voltage levels. At this point the device configuration data stored in the LCA will



```
11111111                             - DUMMY BITS*                              ⎤
0010                                 - PREAMBLE CODE                            ⎥ HEADER
< 24-BIT LENGTH COUNT >              - CONFIGURATION PROGRAM LENGTH             ⎥
1111                                 - DUMMY BITS (4 BITS MINIMUM)              ⎦

0 < DATA FRAME # 001 > 111  ⎫            FOR XC3020                             ⎤
0 < DATA FRAME # 002 > 111  ⎪                                                   ⎥
0 < DATA FRAME # 003 > 111  ⎪            197 CONFIGURATION DATA FRAMES          ⎥ PROGRAM DATA
  .         .          .    ⎬                                                   ⎥
  .         .          .    ⎪            (EACH FRAME CONSISTS OF:               ⎥ REPEATED FOR EACH LOGIC
  .         .          .    ⎪              A START BIT (0)                      ⎥ CELL ARRAY IN A DAISY CHAIN
0 < DATA FRAME # 196 > 111  ⎪              A 71-BIT DATA FIELD                  ⎥
0 < DATA FRAME # 197 > 111  ⎭              THREE STOP BITS                      ⎥
                                                                                ⎥
1111                                 POSTAMBLE CODE (4 BITS MINIMUM)            ⎦
```

*THE LCA DEVICES REQUIRE 4 DUMMY BITS MIN., XACT 2.10 GENERATES 8 DUMMY BITS

1105 06

Device	XC3020	XC3030	XC3042	XC3064	XC3090
Gates	2000	3000	4200	6400	9000
CLBs Row X Col	64 (8 X 8)	100 (10 X 10)	144 (12 X 12)	224 (16 X 14)	320 (20 X 16)
IOBs	64	80	96	120	144
Flip-flops	256	360	480	688	928
Bits per frame (w/ 1 start 3 stop)	75	92	108	140	172
Frames	197	241	285	329	373
Program Data = Bits * Frames + 4 (excludes header)	14779	22176	30784	46064	64160
PROM size (bits) = Program Data + 40 bit Headers	14819	22216	30824	46104	64200

Figure 19 LCA configuration data structure [form Ref. 1]

totally specify its functionality and interconnect. At any time, the LCA configuration memory may be re-programmed thus totally changing the characteristics and functionality of the LCA.

With an understanding of the LCA technology this thesis will now look at an actual design. In Chapter III the design example of SCSI will be defined.

III. PRINCIPLES OF THE SMALL COMPUTER SYSTEM INTERFACE (SCSI)

A. BASIC OPERATION

The purpose of this chapter is to introduce the basic concepts of the Small Computer Systems Interface (SCSI) protocol. The information contained in this chapter is derived from Reference 4. This protocol will be used as the basis for a design example to demonstrate the methodology behind implementation of design into a FPGA. It will also be used as a benchmark to evaluate FPGA technology performance.

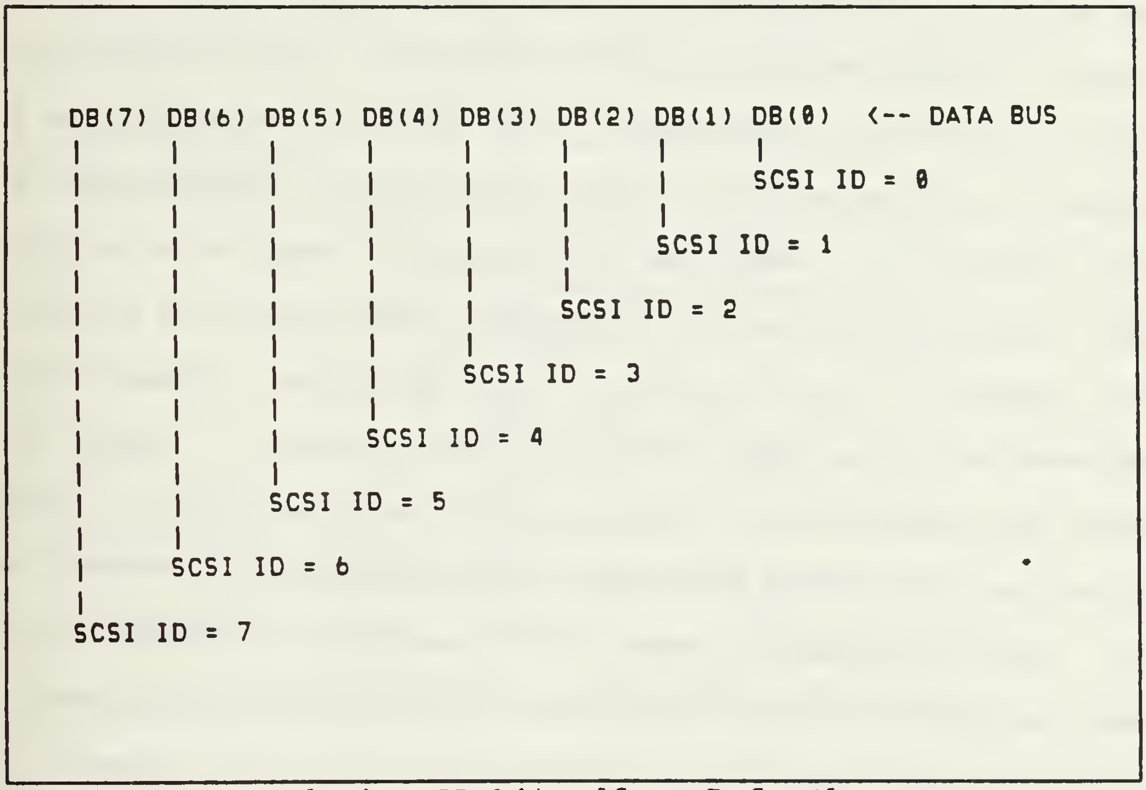


Figure 20 SCSI device ID bits [from Ref. 4]

The SCSI is a protocol that allows asynchronous or synchronous bidirectional communication between two devices. It allows for priority arbitration when more than one device is trying to use the bus. It allows specific target device addressing from the controller. Information is transfered via an asynchronous handshaking protocol. Information is transfered via a common data bus and may take the form of data, control signals or messages. The basic SCSI allows communication between only two SCSI devices at any one time. There is a maximum of eight SCSI devices that can be connected to the SCSI bus. In this thesis the only SCSI of interest is the original asynchronous SCSI standard design. Later SCSI designs such as SCSI-II are not considered. Here, only non-parity systems are considered. Each SCSI device will have a unique SCSI ID which is actually a bit pattern assigned to it (see Figure 20). When two SCSI devices communicate on the bus, one acts as an initiator and the other acts as a target. For example, a host computer (the initiator) transfering information to a disk controller (the target). Figure 21 shows two examples of typical SCSI configurations. Each target may have seven additional SCSI peripherals attached to it. Using extended messages it is possible to address up to 2048 peripheral devices per target. The initiator has control of certain SCSI bus functions while the target has control of the rest. The initiator may arbitrate for the SCSI bus and select a particular target while the target may request the

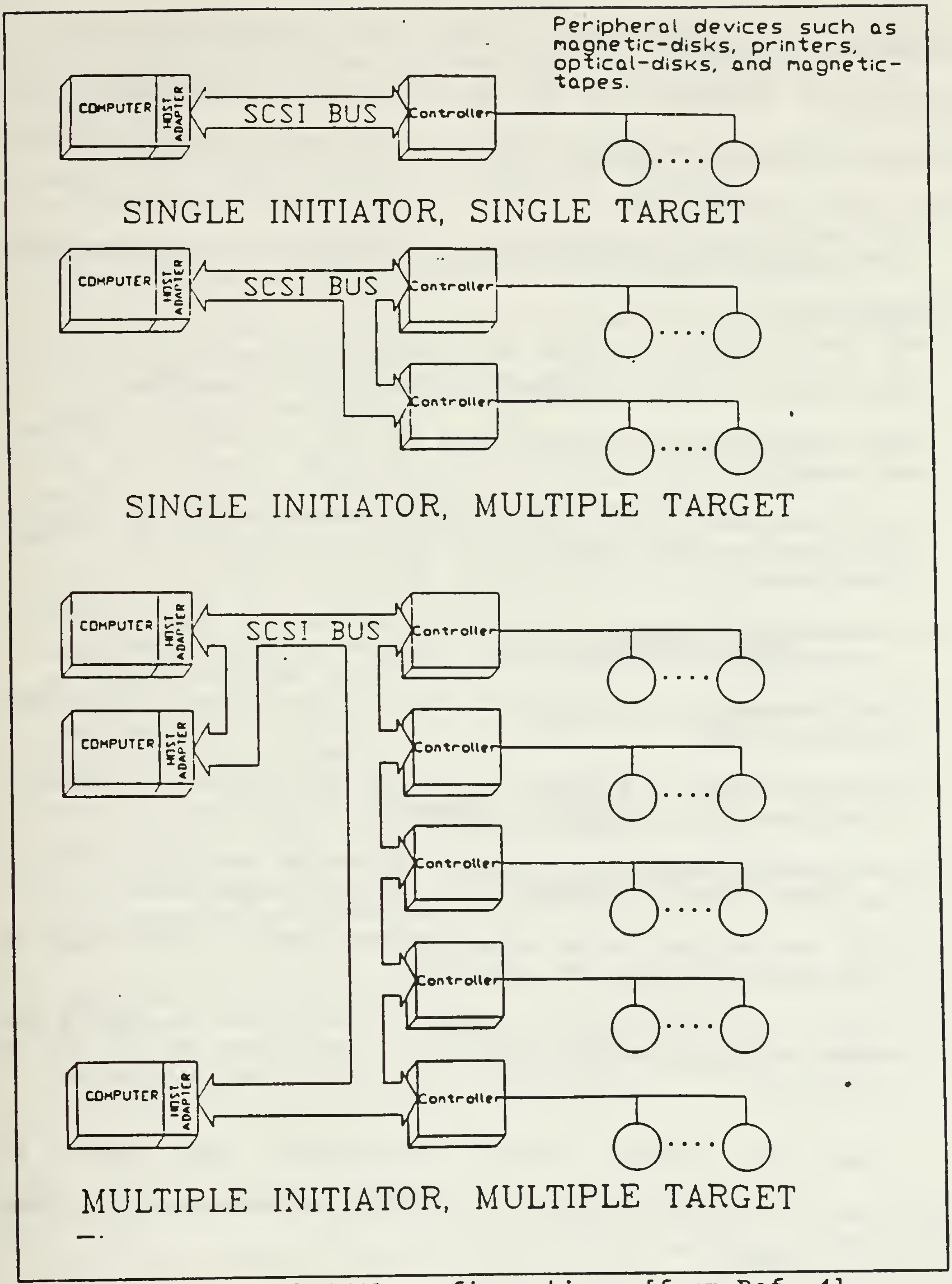

Peripheral devices such as magnetic-disks, printers, optical-disks, and magnetic-tapes.
COMPUTER
HOST ADAPTER
SCSI BUS
Controller
SINGLE INITIATOR, SINGLE TARGET
SINGLE INITIATOR, MULTIPLE TARGET
MULTIPLE INITIATOR, MULTIPLE TARGET

Figure 21 Typical SCSI configurations [from Ref. 4]

transfer of command, data, status or message information. Information transfers on the data bus are asynchronous and utilize a REQ/ACK handshake protocol with each byte transfered requiring a handshake.

There are eighteen SCSI bus signals which are summarized as follows:

- BSY (Busy) indicates that the bus is being used.
- SEL (Select) is a signal used by the initiator to select a target.
- C/D (Control/Data) is a signal driven by a target to indicate whether control or data information is on the data bus.
- I/O (Input/Output) is a signal driven by the target to control data direction with respect to the initiator.
- MSG (Message) is a signal driven by the target during the message phase.
- REQ (Request) is a signal driven by the target to indicate a request for a REQ/ACK handshake.
- ACK (Acknowledge) is a signal driven by the initiator to indicate acknowledgement of a REQ/ACK handshake.
- ATN (Attention) is a signal driven by the initiator to indicate an attention condition.
- DB(0-7,P) (Data Bus) is eight bit data plus a parity bit which make up the SCSI data bus. Use of the parity bit is optional.

There are several timing definitions that should be included in any SCSI design to provide standardization. They are as follows:

- ARBITRATION DELAY (2200 nS) is the minimum time a SCSI device shall wait from the time that it arbitrates for the SCSI bus until it checks to see if it arbitration has been won.

- ASSERTION PERIOD (90 nS) is the minimum time REQ or ACK are asserted during synchronous data transfer.

- BUS CLEAR DELAY (800 nS) in the maximum time allowed for a SCSI device to quit driving all SCSI bus signals.

- BUS FREE DELAY (800 nS) is the minimum time that a SCSI device must wait from the time that it detects that the bus has been free for 400 nS before it can enter the arbitration phase by asserting BSY.

- BUS SET DELAY (1800 nS) is the maximum time for a SCSI device to assert BSY and put it's own SCSI ID bit on the DATA BUS after it enters the bus free phase.

- BUS SETTLE DELAY (400 nS) is the time to wait for the SCSI bus to settle after changing bus control signals.

- CABLE SKEW DELAY (10 nS) is the maximum difference in propagation time between any two SCSI bus signals.

- DESKEW DELAY (45 nS) is the minimum time required to allow for deskewing signals.

- DATA RELEASE DELAY (400 nS) is the maximum time for an initator to shift from sending data to receiving data.

- HOLD TIME (45 nS) is the minimum time that the SCSI device must wait after asserting REQ or ACK prior to changing the data on the SCSI data bus.

- RESET HOLD TIME (25000 nS) is the minimum time RST must be asserted.

- SELECTION ABORT TIME (200000 nS) is the maximum time that a target has to respond to selection.

B. SCSI AS A STATE MACHINE

The SCSI may be looked at as a state machine. Depending on the particular implementation however, it may or may not

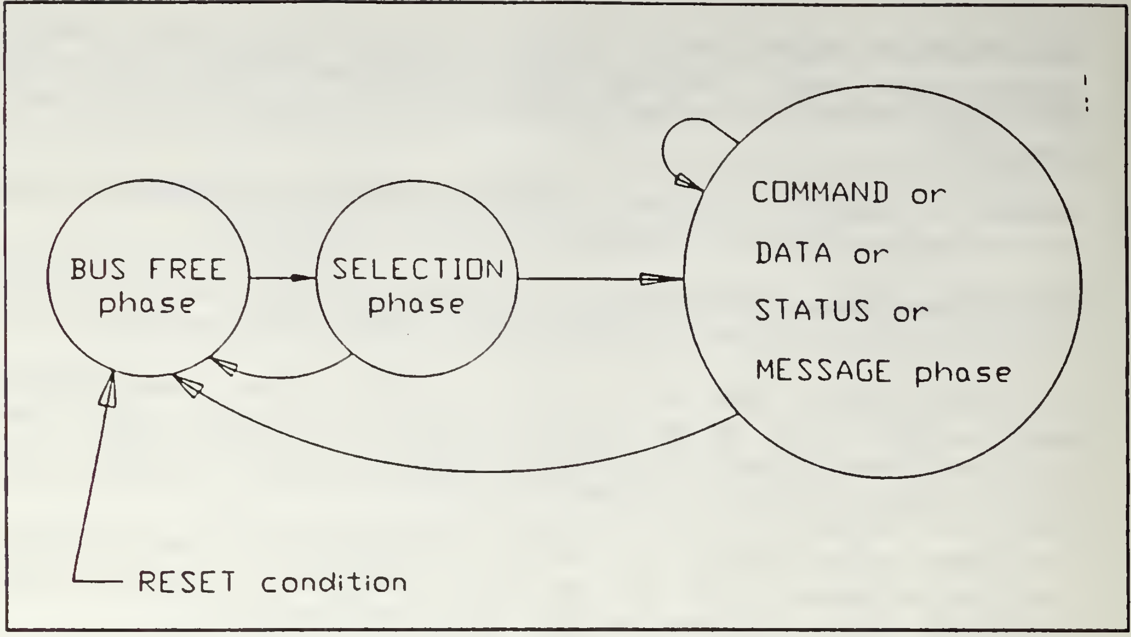


Figure 22 Non-arbitrating SCSI [from Ref. 4]

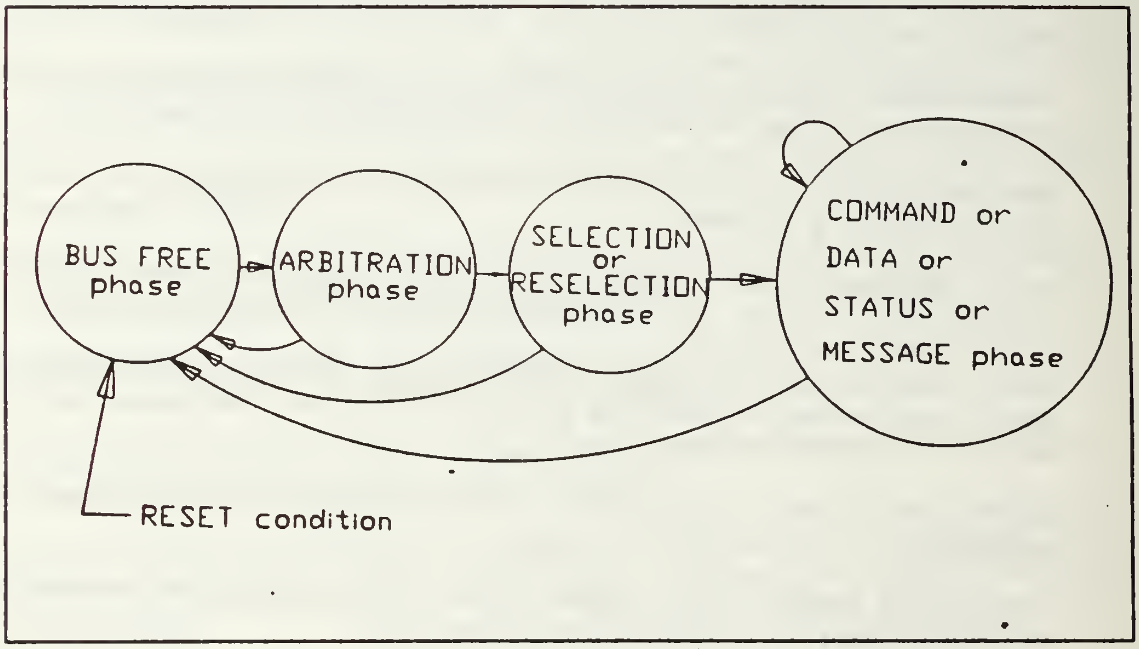

BUS FREE phase
ARBITRATION phase
SELECTION or RESELECTION phase
COMMAND or DATA or STATUS or MESSAGE phase
RESET condition

Figure 23 Arbitrating SCSI [from Ref. 4]

contain all of the states. Figure 22 shows a three state non-arbitrating SCSI while Figure 23 shows a four state arbitrating SCSI. The implementation chosen for use in this thesis is one with full arbitration capabilities. The basic

SCSI design involves a state machine with four states which are:

- BUS FREE PHASE
- ARBITRATION PHASE
- SELECTION PHASE
- INFORMATION TRANSFER PHASE

1. Bus free phase

The SCSI device is in this phase when no other SCSI device is using the bus and the bus is available. To enter this phase, BSY and SEL must both be false for at least one bus settle delay (400 ns). It is also required that any other SCSI device that was driving the bus must reléase the bus within a bus clear delay (800 ns) after BSY and SEL have been false for a bus settle delay (400 ns).

2. Arbitration Phase

This phase allows one SCSI device to take control of the bus even when another device tries to gain control. The SCSI ID bit is the vehicle for accomplishing this. Since each SCSI device has a unique SCSI ID bit the device with the bit in the most significant position will win the arbitration and the other device request will be masked. The device that loses arbitration goes back to wait for the bus free phase to be detected.

When a SCSI device detects bus free, it waits a bus free delay (800 ns) and then asserts BSY as well as puts its SCSI ID bit on the SCSI data bus. After an arbitration delay (2200 ns) from the assertion of BSY to ensure that any other device that wants to compete for the bus have had the opportunity to do so, the SCSI device will read the data bus to determine whether or not there exists a device with a higher order SCSI ID bit. If there is not, the device has won the arbitration and it asserts SEL. If there exists a higher SCSI ID bit on the data bus, the device loses the arbitration, releases BSY and its own SCSI ID within a bus clear delay (800 ns) and waits for bus free.

3. Selection Phase

In this phase the initiator will select a target to either write to or read from. At the beginning of this phase, BSY and SEL have both been asserted by the initiator and a bus clear delay plus a bus settle delay have passed (1200 ns). The SCSI device becomes the initiator by releasing I/O which is driven by the target. The initiator sets the data bus to the logical OR of its own SCSI ID and the target SCSI ID. The initiator waits 2 deskew delays (90 ns) and releases BSY. The initiator then waits at least a bus settle delay (400 ns) for a response from the target. The target determines that it has been selected when SEL and its own SCSI ID bit which was placed on the data bus by the initiator are true and BSY and

I/O are false. If, upon examining the data bus, a SCSI device determines that it is a target of some initiating SCSI device, it asserts BSY. After 2 deskew delays (90 ns) the initiator releases SEL.

4. Information transfer phase

This is the state where the SCSI normally operates. During this phase data, commands, status, and messages are transfered between the initiator and the target. The target drives C/D, I/O and MSG to setup different types of information transfer. The target will set I/O to true to transfer information from the target to the initiator and will set I/O to false to transfer information from the initiator to the target. To send information from the target to the initiator, the target drives the data bus. The target starts by setting I/O true. The target asserts REQ and after one deskew delay plus a cable skew delay (55 ns) to ensure valid data, the initiator reads the data bus. The initiator then asserts ACK. When ACK is true at the target, the target may change or release the data bus. The target then negates REQ and the initiator negates ACK. This process is then repeated byte by byte until no further information is to be transferred. To transfer information from the initiator to the target, the target sets I/O to false and asserts REQ. After one deskew delay plus a cable skew delay (55 ns) the initiator asserts ACK to let the target know that there is valid data.

The target then reads the data bus and then negates REQ to let the initiator know that it can change the data bus. After REQ is false at the initiator, the initiator may change the data bus. The initiator then negates ACK to let the target know that it can send another byte if it is ready. This process is repeated until there is no more information to be sent.

There are four types of information that can be sent over the SCSI bus. They are:

- DATA which can go in either direction
- COMMAND which goes from initiator to target
- STATUS which goes from target to initiator
- MESSAGE which can go in either direction

The type of information that is sent over the SCSI bus is a function of how the I/O, C/D and MSG lines are driven. In this thesis we will be using a SCSI implementation that sends data only.

The SCSI is used here as a design example for the implementation of design into an FPGA. It was chosen because the design performance of such a device using other technologies is known providing a good comparison of FPGA design performance with other design implementation methods. The SCSI possesses characteristics that would lend itself as a test case for FPGA design implementation.

IV. THE DESIGN PROCESS USING FIELD PROGRAMMABLE GATE ARRAYS

A. OVERVIEW

Information contained in this chapter is derived from Reference 1. In this chapter, the typical design cycle using the XILINX Development System to implement design into FPGAs is discussed in detail. The purpose of this discussion is to provide familiarization with the process so that the actual design example in Chapter V may be presented effectively. Implementation using FPGAs allows the logic designer to realize small to medium sized designs that have in the past been relegated to implementation into custom (and expensive) Application Specific Integrated Circuits (ASICs). The FPGA allows the designer to inplement designs with the flexibility of being able to modify those designs quickly and inexpensively. This design implementation into method allows the designer to come up with a logic design using industry standard schematic capture packages or design description languages. Through the use of software conversion, those designs can be mapped into a fully programmable array of logic cells. Once this mapping is complete, it can be routed (interconnected) in such a way to optimize circuit performance. Once the design has been mapped and routed, it must be converted to a bit stream or a PROM file of configuration data for actually programming the LCA. One of the main issues that will be looked at in this thesis is the issue of design verification both prior to and after routing. In the past, timing simulation could be run on designs that had been described via schematic capture. The issue with FPGAs is what effect does LCA routing have on circuit performance. This will require backannotating the routed LCA design to incorporate routing delays into the timing simulation. This is a major issue that this thesis is trying to resolve. Additionally, the question as to whether or not hand routing will be required to optimize design performance either from the aspect of meeting the required timing considerations or to accomodate the design into the LCA of a chip. Router constraint issues will also be looked at. The question of what needs to be done to implement previously tested designs in a modular fashion into FPGAs will also be examined. Design implementation will be conducted using a personal computer as well as an engineering workstation. The differences and advantages of the design platforms will also be studied.

B. DESIGN CYCLE ON A PERSONAL COMPUTER

The design platform used was an IBM Personal System/2 Model 50 with 3 MB of RAM and a 20 MB Hard Drive. This system was adequate for small efforts such as training and familiarization with the design process. Memory upgrade is required however, for more involved designs. Table II shows the memory requirements for various design and device

Table II LCA DESIGN MEMORY REQUIREMENTS [from Ref. 1]

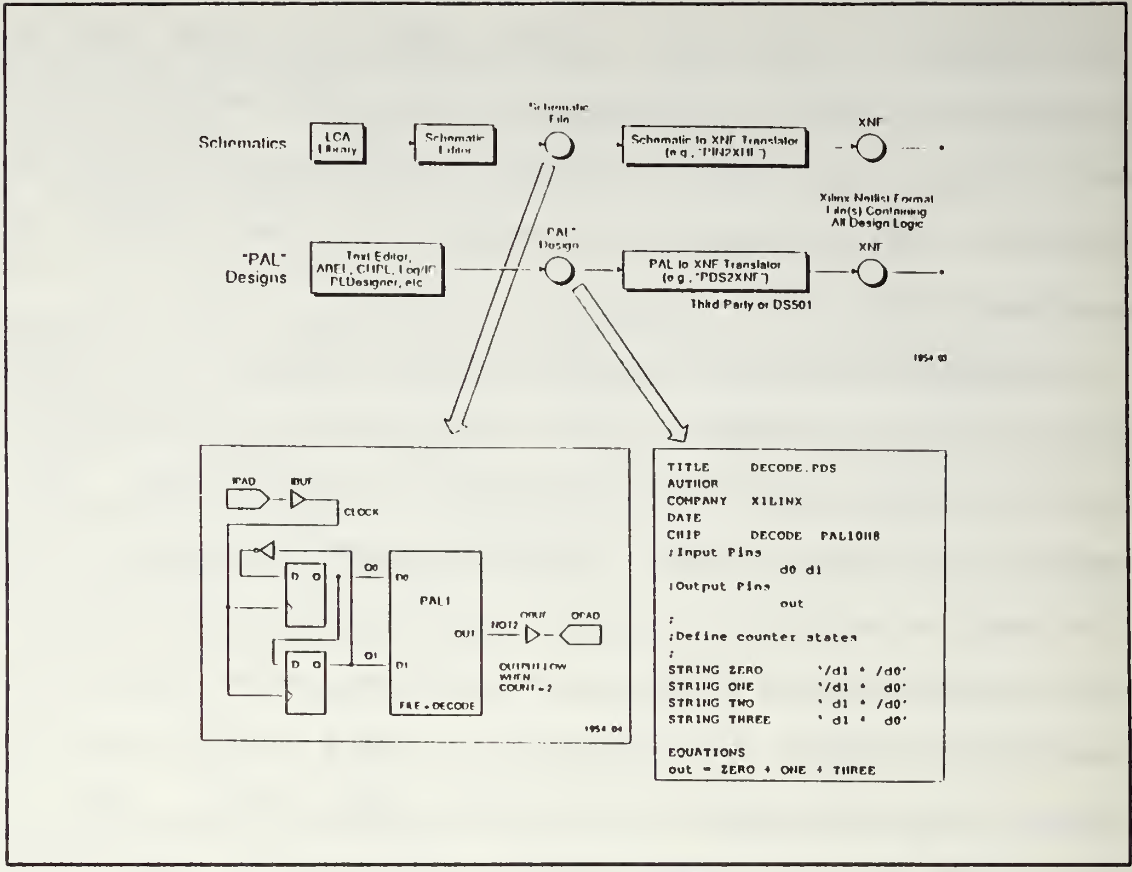
2000 gates	2064, 2018, & 3020 LCA	2.50 Mbytes
3000 gates	3030 LCA	3.25 Mbytes
4200 gates	3042 LCA	4.00 Mbytes
6400 gates	3064 LCA	5.25 Mbytes
9000 gates	3090 LCA	6.50 Mbytes

Note: Other resident programs not included, 0.5 MB less without XDM.

complexities. The beginning of the design process involves the initial layout of the design. Two methods are available to support this. The two design entry methods supported by the XILINX Development System are schematic capture and design description language entry. Most popular schematic capture packages are supported in this development environment. Schematic capture is used to layout the design while a design description language is used to describe a design and place that description into a Programmable Array Logic (PAL) circuit. The XILINX Development System supports either or both of these methods. The schematic capture method utilizes a standard library of parts in conjunction with a schematic editor to build a schematic file whereas the design description language approach uses a text editor to describe the PAL design. It should be noted that design description entries for PAL may be included in schematic designs, but a "FILE = " must be included in the PAL symbol to cross reference the description language text file to be incorporated into the design. In this thesis, FUTURENET was used as the tool for schematic capture. At this point timing simulation of the design is prudent prior to taking the design into the XILINX Development System. For instance CADAT could be used to simulate the design functionality. However, due to the difficulty of use and unreliability of that simulator, the choice was made to reconstruct the design schematics in the Mentor Graphics Development System with NETED and run timing simulation in QUICKSIM to verify design functionality. Once the design verification was completed satisfactorily, then the FUTURENET design schematics could be implemented into the FPGA via the XILINX development system on the PC.

The heart of the XILINX Development System is the XILINX Design Manager (XDM). The XDM is a shell that manages all software functions. Unfortunately, it took up 0.5 MB of user memory. Due to limited memory resources XDM could not be run concurrently with the XACT program or APR program when a user tries to place and route designs of 2000 gates or larger. The XACT program run by the XACT Executive is used to make the configuration bit file and allow editing of LCA files manually. The APR program is used to automatically place and route designs in the LCA. In the first case, XDM must be suspended and XACT run separately while in the later case APR must be run from DOS. These inconveniences could be eliminated by upgrading the amount of user memory in accordance with Table II.

The first step which is necessary to implement the design is to convert all schematic file (which could be hierarchical) and PAL design files into XILINX NETLIST FILES (XNF). With regards to schematic files, FUTURENET interface outputs a PIN file that is converted to an XNF file by the PIN2XNF program while the PAL design PDS file is converted to an XNF file by the PDS2XNF program (see Figure 24). The result is that all design files have been put into the XNF format. The PAL design XNF file is additionally optimized by the XNFOPT program to reduce the combinational logic contained in the design so that it will fit better in the LCA. The result is that an optimized XNF file is generated from the PAL design file (see Figure 25). The next step is to map the XNF files into Logic Cell Array (LCA) Configurable Logic Blocks (CLBs) and Input/Output Blocks (IOBs) (see Figure 25). Once all of the XNF files have been mapped they are merged into one map file by XNFMERGE (see Figure 26). The final map file is then put into the LCA file by the MAP2LCA program (see Figure 27). At this point the LCA file is "unplaced" and "unrouted", which means that there have been no IOBs or CLBs chosen and no interconnect performed. The next step is to place and route


Schematics
LCA Library
Schematic Editor
Schematic File
Schematic to XNF Translator (e.g., "PIN2XNF")
XNF
Xilinx Netlist Format File(s) Containing All Design Logic
"PAL" Designs
Text Editor, ABEL, CUPL, Log/IC PLDesigner, etc
"PAL" Design
PAL to XNF Translator (e.g., "PDS2XNF")
Third Party or DS501
XNF
1954 03
IPAD
IBUF
CLOCK
D Q
D Q
Q0
Q1
D0
D1
PAL1
OUT
NOT2
OBUF
OPAD
OUTPUT LOW WHEN COUNT = 2
FILE = DECODE
1954 04

```
TITLE      DECODE.PDS
AUTHOR
COMPANY    XILINX
DATE
CHIP       DECODE   PAL10H8
;Input Pins
              d0 d1
;Output Pins
              out
;
;Define counter states
;
STRING ZERO        '/d1 * /d0'
STRING ONE         '/d1 *  d0'
STRING TWO         ' d1 * /d0'
STRING THREE       ' d1 *  d0'

EQUATIONS
out = ZERO + ONE + THREE
```

Figure 24 Design Entry to XNF translation [from Ref. 1]

the design. There are two ways available to place and route. The first is an automatic place and route which uses an annealing algorithm to get the best performance from the design. There are three routing options available in the automatic router for the best performance:

- Use router directly
- Use a constraint file
- Use a guide file

In using the router directly, there are three router options available: R1, R2 and R3. R3 which is delay driven is the

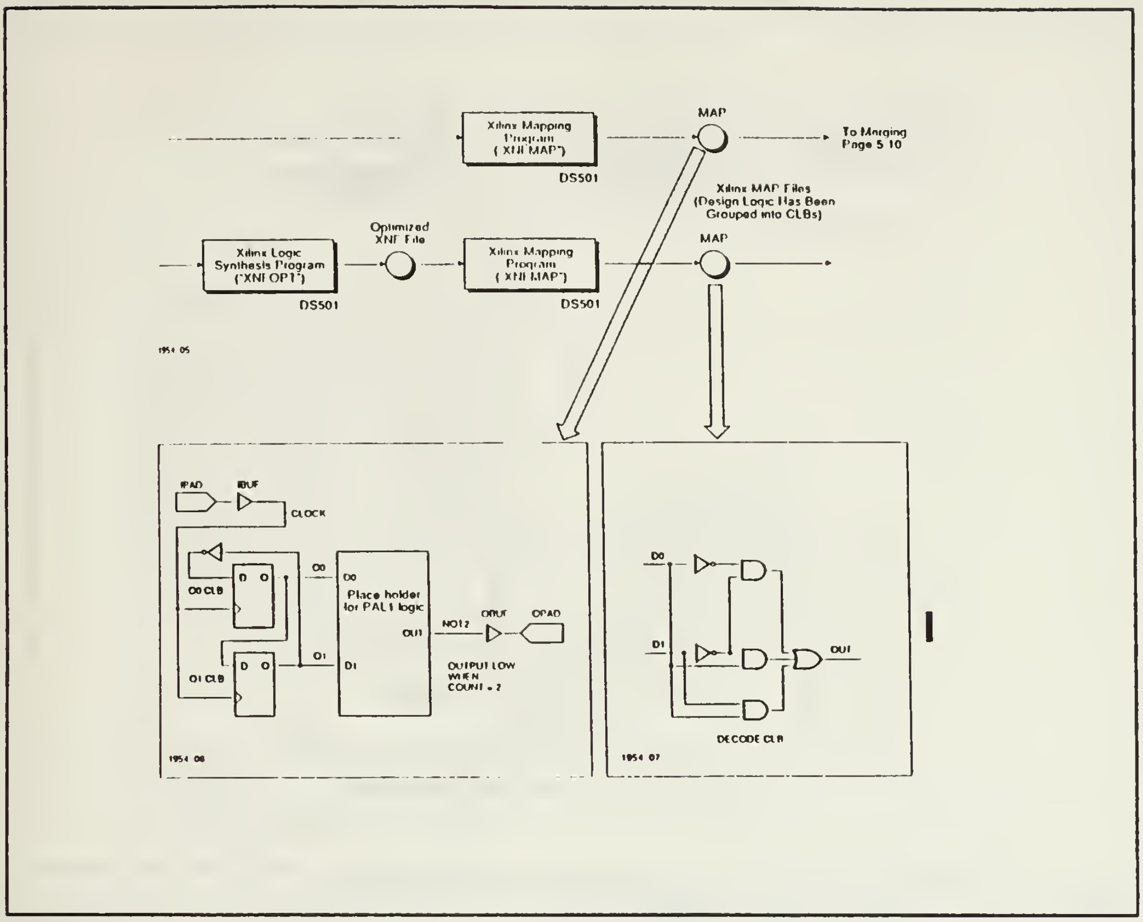
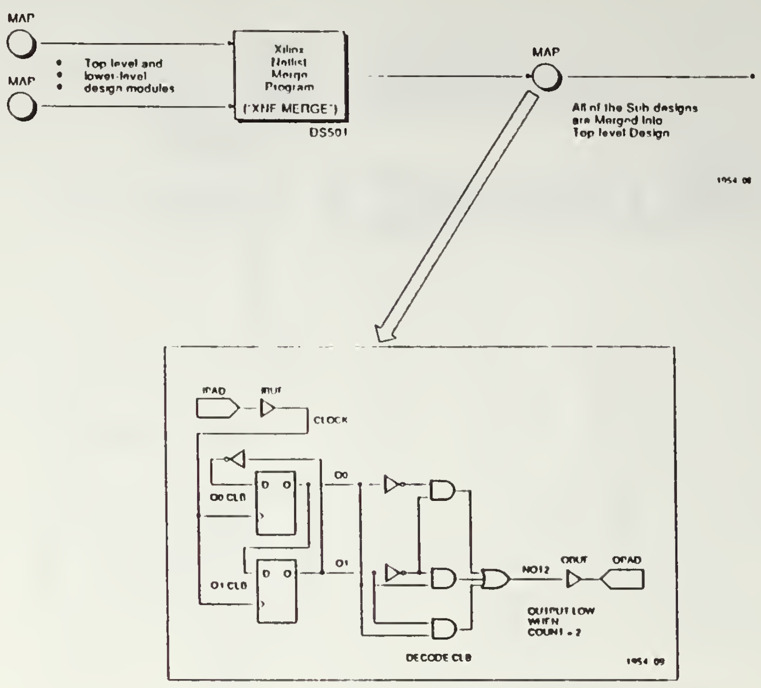

Xilinx Mapping Program ("XNFMAP")
DS501
MAP
To Merging Page 5 10
Xilinx MAP Files (Design Logic Has Been Grouped into CLBs)
Xilinx Logic Synthesis Program ("XNFOPT")
DS501
Optimized XNF File
Xilinx Mapping Program ("XNFMAP")
DS501
MAP
1954 05
IPAD
IBUF
CLOCK
D O
O0 CLB
D O
O1 CLB
O0
O1
D0
Place holder for PAL1 logic
OUT
D1
NO12
OBUF
OPAD
OUTPUT LOW WHEN COUNT = 2
1954 06
D0
D1
OUT
DECODE CLB
1954 07

Figure 25 Optimization and mapping [from Ref. 1]

slowest of the three but results in the lowest delays and unroutes. R1 is the fastest router of the three. The use of the constraint file option allows the user to supply a user constraint file "filename.cst" to a provide direction to the router. This file is used in addition to any schematic constraint file "filename.scp" that may exist as a result of the processing of the schematic by the XILINX system. The user constraint file overrides any schematic constraint file if possible. The -C option must be selected when running APR to utilize this option. The guide file option allows the user


MAP
MAP
Top level and lower-level design modules
Xilinx Netlist Merge Program
("XNF MERGE")
DS501
MAP
All of the Sub designs are Merged Into Top level Design
1954 08
IPAD
IBUF
CLOCK
Q0
Q0 CLB
Q1
Q1 CLB
D Q
NO12
OBUF
OPAD
OUTPUT LOW WHEN COUNT = 2
DECODE CLB
1954 09

The merged design contains the CLBs and IOBs for the entire design.

Figure 26 Merging of mapped files [from Ref. 1]

to incorporate design changes but use the placement and routing of the old design. This option utilizes a block, pin, and net name matching scheme. If a match occurs between the new design and the guide file, the guide file placement and routing information is used. To use the guide file, the -G option must be used when APR is invoked. Manual routing can also be performed using the EDITLCA program from the XACT design editor (see Figure 28).

Once a placed and routed LCA file has been generated, the MAKBITS program (see Figure 29) which is run from the XACT Executive will be used to generate the configuration bitstream

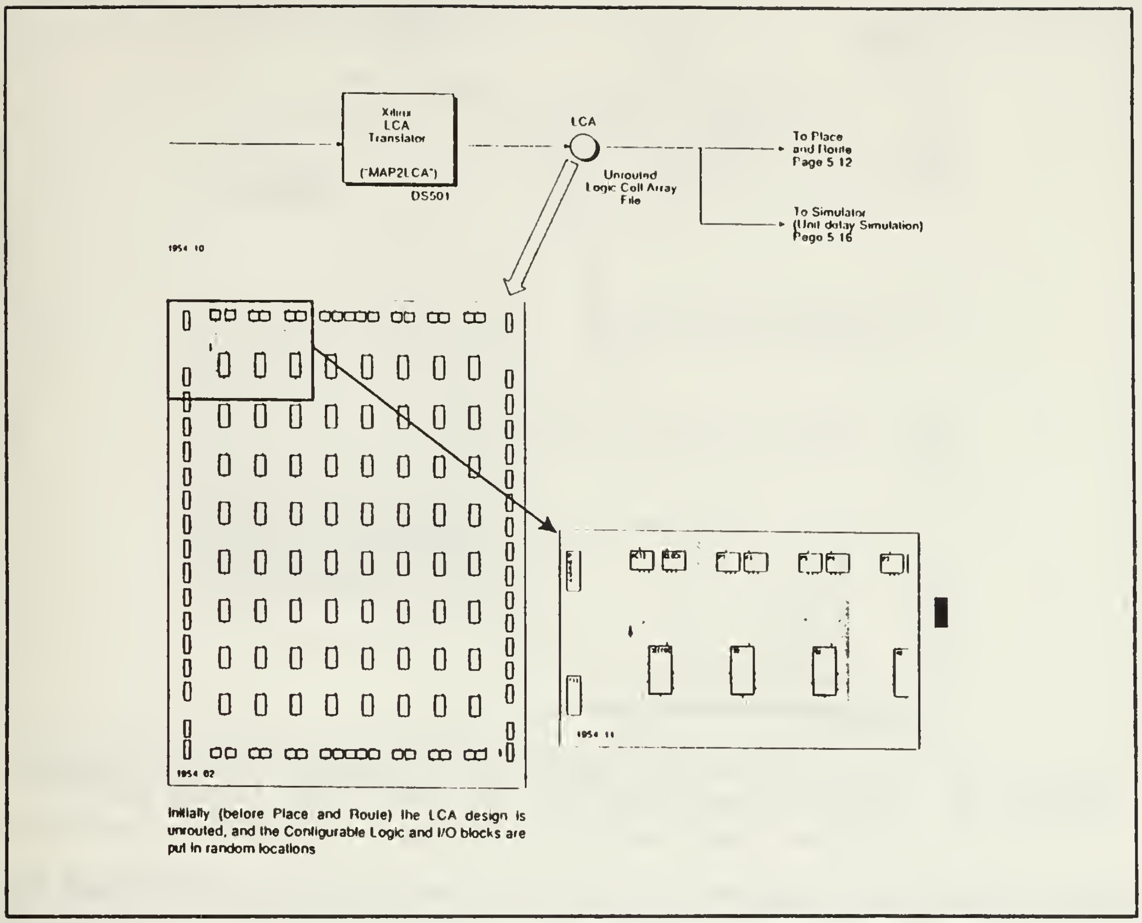
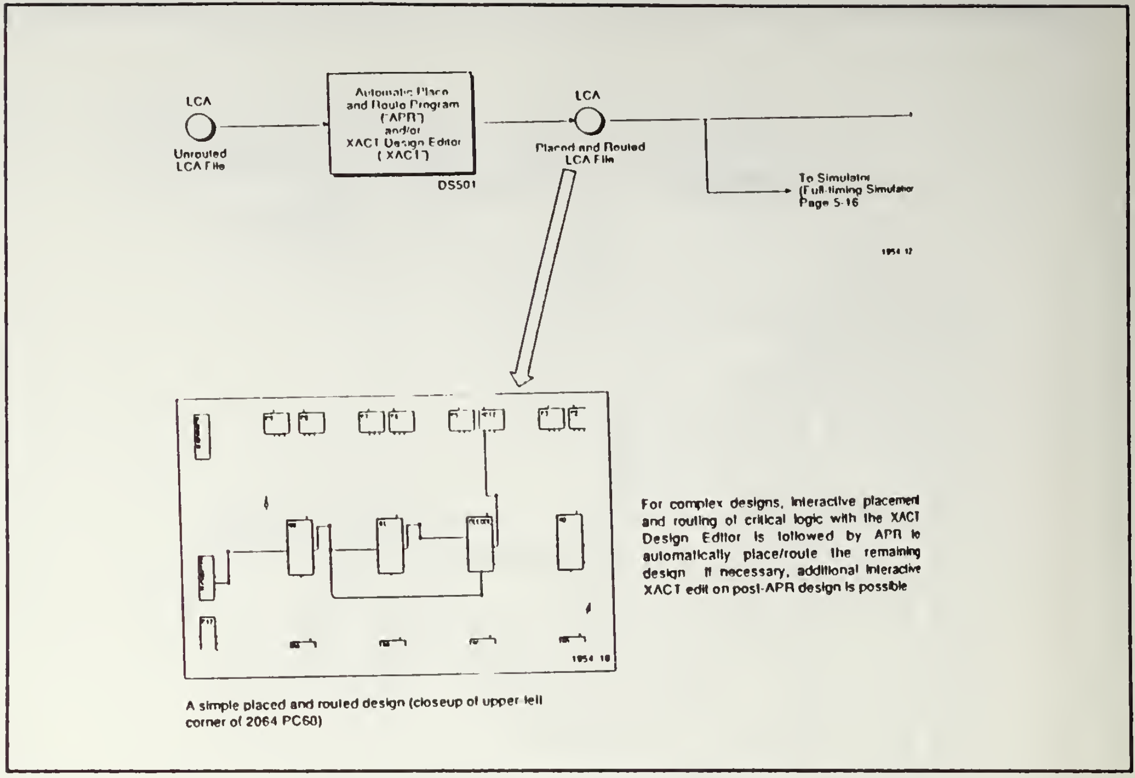


Figure 27 Translating to an LCA file [from Ref. 1]

which can be downloaded into the LCA to configure it. The bit file can also be used to make a programming file to program an EPROM (see Figure 30). An EPROM is a good choice because it is inexpensive and reuseable and lends itself ideally to prototype designs. The XILINX Development System supports three programming formats:

- MCS86 - Intel MCS-86 Hexadecimal Object
- EXORMAX - Motorola Exormax
- TEKHEX - Tektronix Hexadecimal


LCA
Unrouted LCA File
Automatic Place and Route Program (APR) and/or XACT Design Editor (XACT)
DS501
LCA
Placed and Routed LCA File
To Simulator (Full-timing Simulator Page 5-16
1954 12
1954 10

For complex designs, interactive placement and routing of critical logic with the XACT Design Editor is followed by APR to automatically place/route the remaining design. If necessary, additional interactive XACT edit on post-APR design is possible.

A simple placed and routed design (closeup of upper left corner of 2064 PC68)

Figure 28 Placing and routing the LCA file [from Ref.1]

These formats are industry standards and can be programmed on very inexpensive PC based EPROM programmers.

The next issue that needs to be examined with regard to implementation is how will the design perform once it has been placed and routed into an LCA. One method is to run the program LCA2XNF on the routed LCA file. This produces an XNF file which contains routing delay information (see Figures 31 and 32). This XNF file can then be converted to a simulator netlist for use in some simulator such as CADAT or SILOS. Since it was already determined that CADAT was unreliable and difficult to use and SILOS was not available, this back-annotation problem will be solved at the engineering

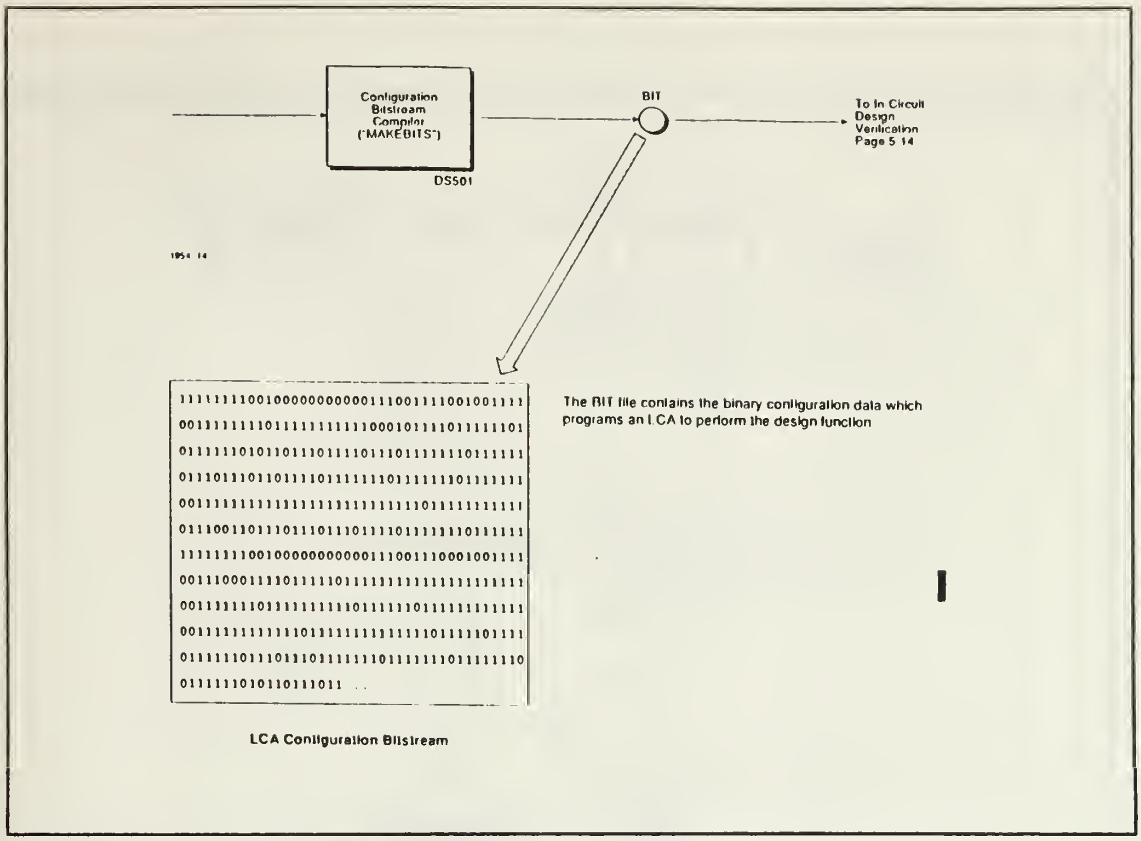

Configuration Bitstream Compiler ("MAKEBITS")
DS501
BIT
To In Circuit Design Verification Page 5 14
The BIT file contains the binary configuration data which programs an LCA to perform the design function
LCA Configuration Bitstream

Figure 29 Bitstream generation [from Ref. 1]

workstation level. Another problem that was not resolved at all was the problem of real-time in-circuit verification as shown in Figure 30. XILINX has an XACTOR real-time in-circuit verifier. It is rather expensive and is unnecessary for this thesis, but it could be purchased at some time in the future. Additionally, there are several third-party vendors that sell verifiers and prototyping boards for use in design development with FPGAs. As an alternative a simple prototype board was developed to attempt to do rudimentary testing of FPGA designs (see Figure 33). The board supports an XC3020 FPGA operating in Master Parallel Low mode with an 8KB or larger EPROM. To use the board with another type of LCA will require a minimal

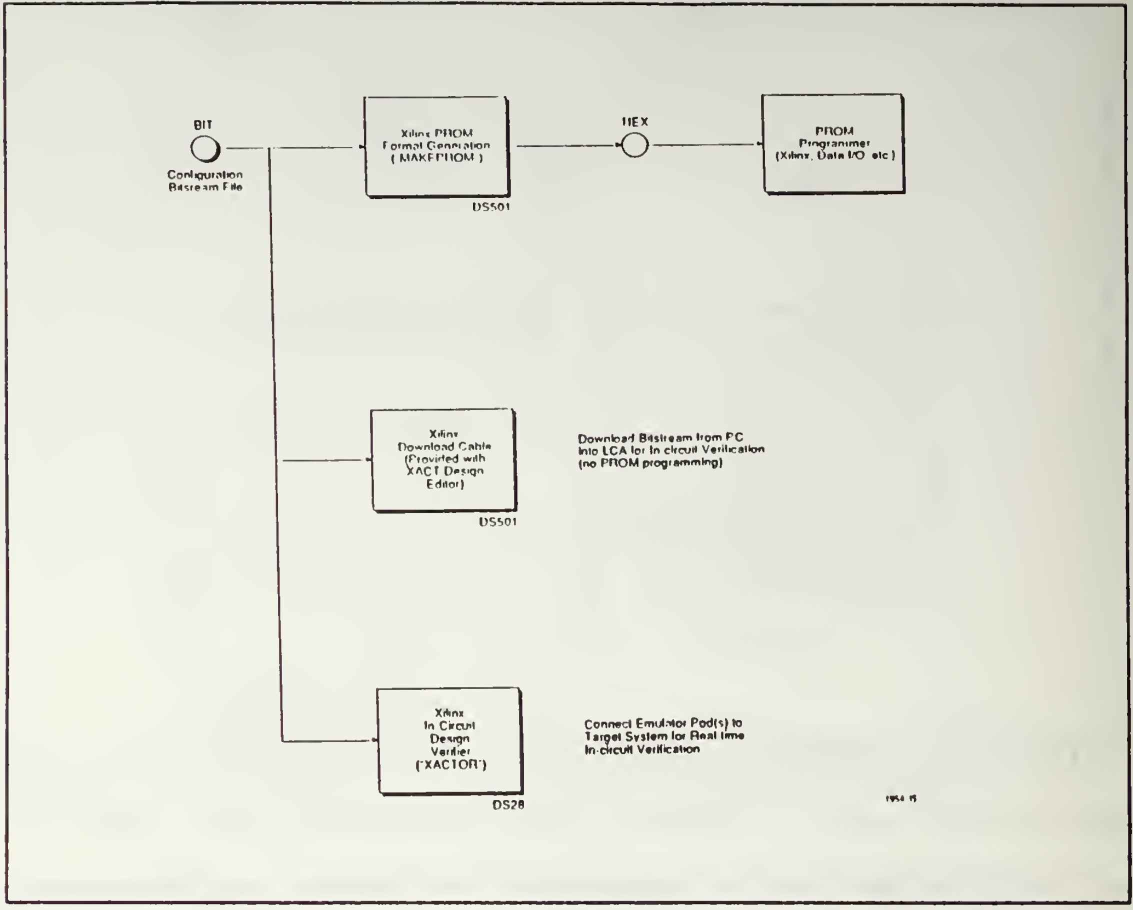

BIT
Configuration Bitstream File
Xilinx PROM Format Generation (MAKEPROM)
DS501
HEX
PROM Programmer (Xilinx, Data I/O etc)
Xilinx Download Cable (Provided with XACT Design Editor)
DS501
Download Bitstream from PC into LCA for In circuit Verification (no PROM programming)
Xilinx In Circuit Design Verifier ("XACTOR")
DS28
Connect Emulator Pod(s) to Target System for Real time In-circuit Verification

Figure 30 Real-time in circuit verification [from Ref. 1]

amount of redesign. Additionally, a demo board was provided by XILINX that served to verify the development system installation. Several sample designs (e.g., a counter) were processed. The counter was downloaded into the configuration memory as well as programmed into an EPROM. LCA configuration and operation were observed to be correct from a macroscopic viewpoint.

Figure 34 shows the overall big picture of the design process on a PC. There are basically four elements:

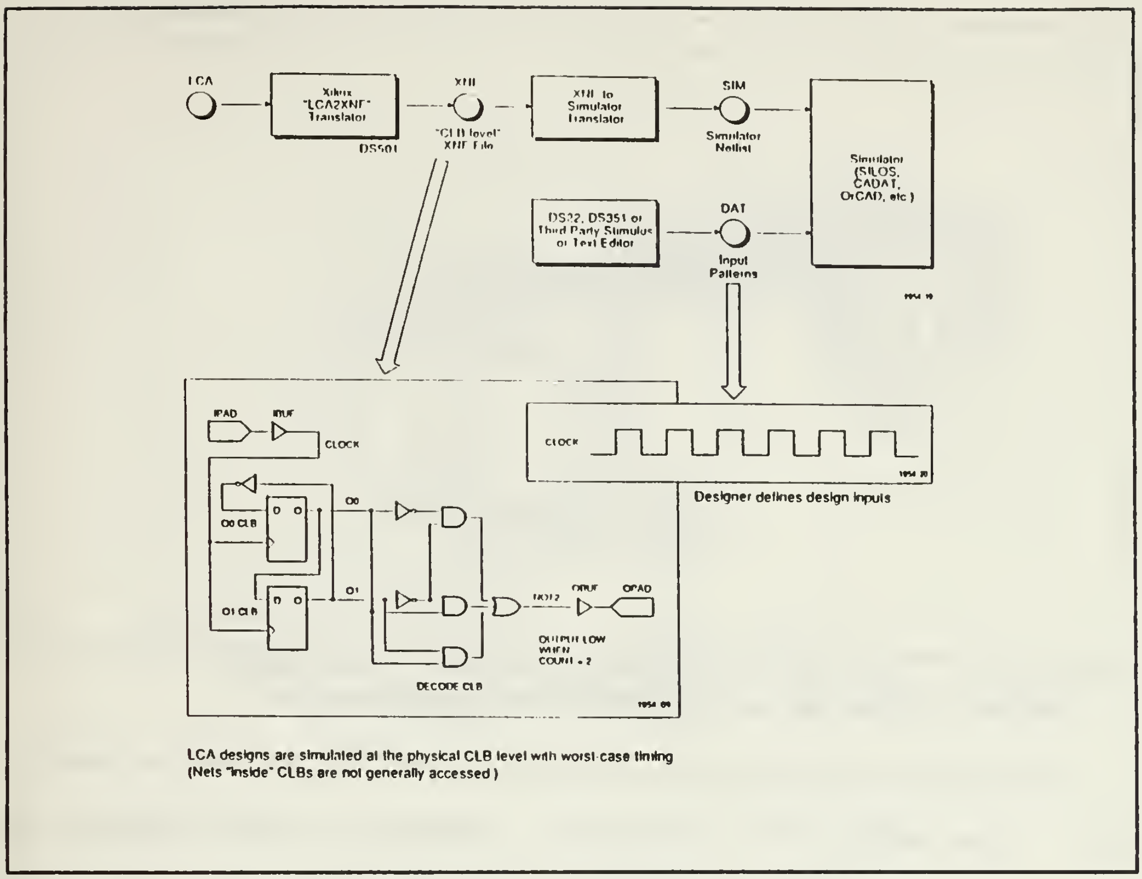

LCA
Xilinx "LCA2XNF" Translator
DS501
XNF
"CLB level" XNF File
XNF to Simulator Translator
SIM
Simulator Netlist
Simulator (SILOS, CADAT, OrCAD, etc.)
DS22, DS351 or Third Party Stimulus or Text Editor
DAT
Input Patterns
IPAD
IBUF
CLOCK
Q0
Q0 CLB
Q1
Q1 CLB
D Q
OBUF
OPAD
OUTPUT LOW WHEN COUNT = 2
DECODE CLB
CLOCK
Designer defines design inputs
LCA designs are simulated at the physical CLB level with worst-case timing (Nets "inside" CLBs are not generally accessed.)

Figure 31 LCA backannotation [from Ref. 1]

- Design Entry
- Design Implementation
- Hardware Circuit Implementation and Verification
- Logic Simulation

On the PC we currently have the ability to do design entry and design implementation. The logic simulation is reserved for the engineering workstation.

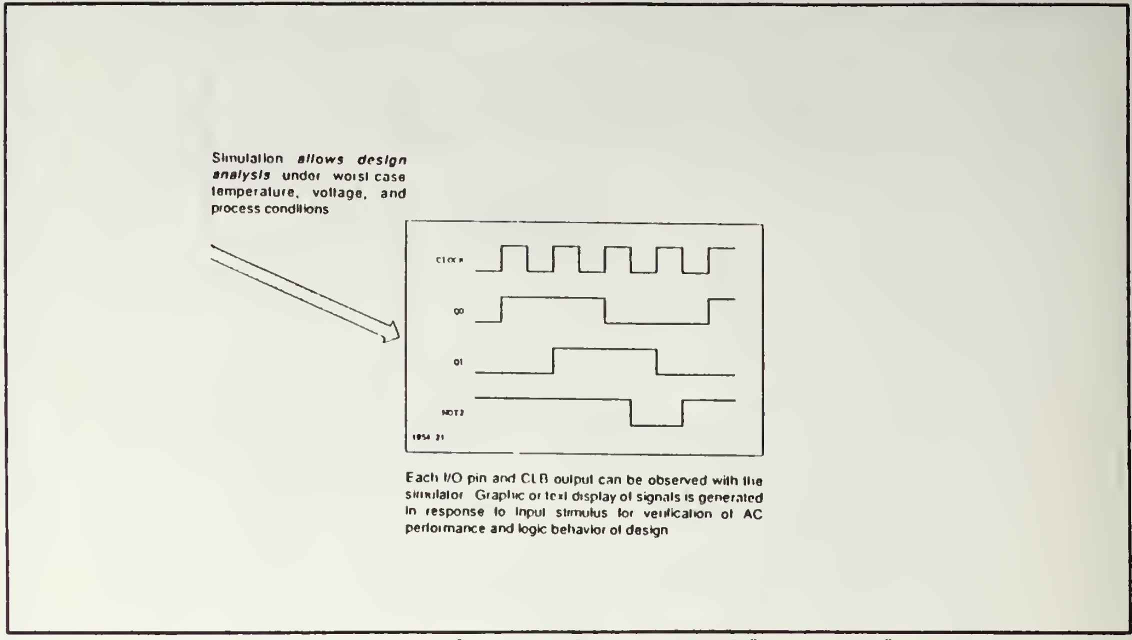

Simulation *allows design analysis* under worst case temperature, voltage, and process conditions
CLOCK
Q0
Q1
NOT2
1954 21
Each I/O pin and CLB output can be observed with the simulator Graphic or text display of signals is generated in response to input stimulus for verification of AC performance and logic behavior of design

Figure 32 Simulation timing diagram [from Ref. 1]

C. DESIGN CYCLE ON AN ENGINEERING WORKSTATION

The design platform used for this variation was an Apollo 4000. The means for schematic capture of the design was Mentor Graphics NETED. This is similiar to what was used in verifying designs on the PC. The design process is virtually identical to that on the PC. The overall design path is shown in Figure 34. In this case however we have installed not only the design entry and implementation tools, but also the simulator. The overall functioning of the various XILINX programs is the same as those on a PC. However, the implementation is somewhat different. The file structure is now UNIX-based vice MS-DOS based. This can cause considerable confusion and various error messages usually related to the ability to find files. It must be understood that what is happening is that operations are occuring across the MENTOR-

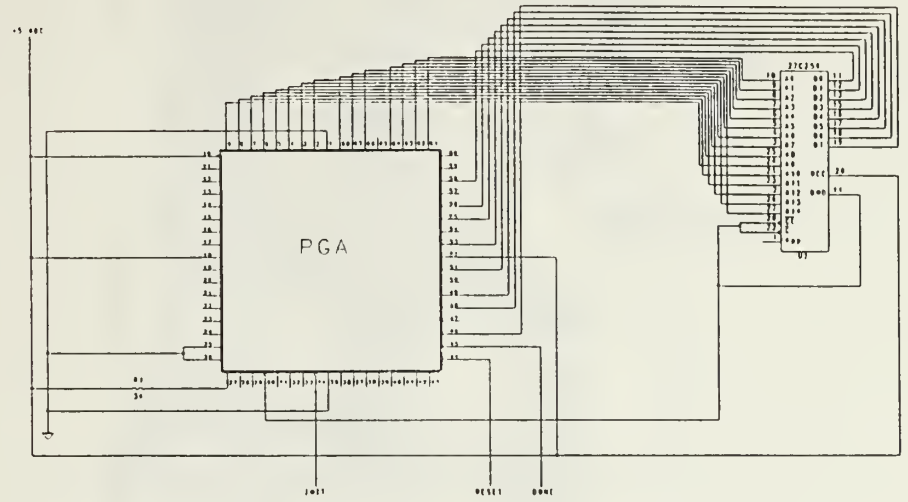


Figure 33 XC3020 prototype board

XILINX interface. It is important to realize that when MENTOR tools are being used, files in a UNIX directory are being accessed. On the other hand, the XILINX tools are based on direct file accesses of the current working directory just like in the MS-DOS environment. As a result, continuing care must be taken in the design process to ensure you have access to the files you need. The XILINX Development System has the characteristic of calling various subroutines from the Mentor portion of the software. It is therefore crucial that all authorization lists for both parts of the system (MENTOR and XILINX) be kept up-to-date or else the whole system will be

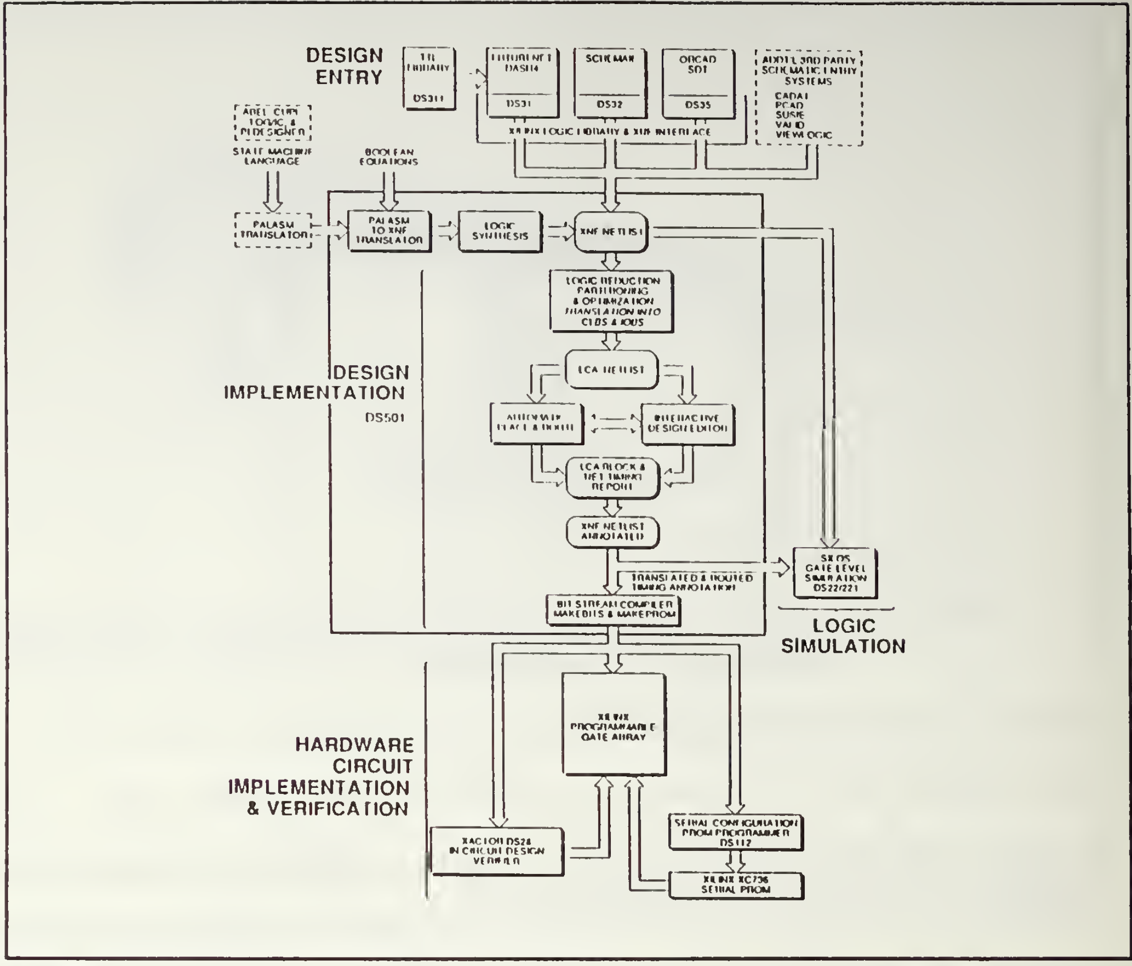

DESIGN ENTRY
TTL LIBRARY DS311
FUTURENET DASH4 DS31
SCHEMA DS32
ORCAD SDT DS35
ADD'L 3RD PARTY SCHEMATIC ENTRY SYSTEMS CADAT PCAD SUSIE VALID VIEWLOGIC
XILINX LOGIC LIBRARY & XNF INTERFACE
ABEL CUPL LOGIC & PLDESIGNER
STATE MACHINE LANGUAGE
BOOLEAN EQUATIONS
PALASM TRANSLATOR
PALASM TO XNF TRANSLATOR
LOGIC SYNTHESIS
XNF NETLIST
LOGIC REDUCTION PARTITIONING & OPTIMIZATION TRANSLATION INTO CLBS & IOBS
DESIGN IMPLEMENTATION
DS501
LCA NETLIST
AUTOMATIC PLACE & ROUTE
INTERACTIVE DESIGN EDITOR
LCA BLOCK & NET TIMING REPORT
XNF NETLIST ANNOTATED
TRANSLATED & ROUTED TIMING ANNOTATION
SILOS GATE LEVEL SIMULATION DS22/221
BIT STREAM COMPILER MAKEBITS & MAKEPROM
LOGIC SIMULATION
XILINX PROGRAMMABLE GATE ARRAY
HARDWARE CIRCUIT IMPLEMENTATION & VERIFICATION
XACTOR DS24 IN CIRCUIT DESIGN VERIFIER
SERIAL CONFIGURATION PROM PROGRAMMER DS112
XILINX XC736 SERIAL PROM

Figure 34 Apollo Workstation Design path [from Ref. 1]

rendered unusable.

The design process starts by using NETED to capture the design schematically. PAL design files are allowed just like on the PC design platform. When using NETED in conjunction with the XILINX development system a different library will be used. LCA_NETED is invoked which uses NETED with the lca_lib library instead of the normal gen_lib library. The lca_lib consists of the XILINX macros and parts for use in designs. The development system allows the construction of macros for

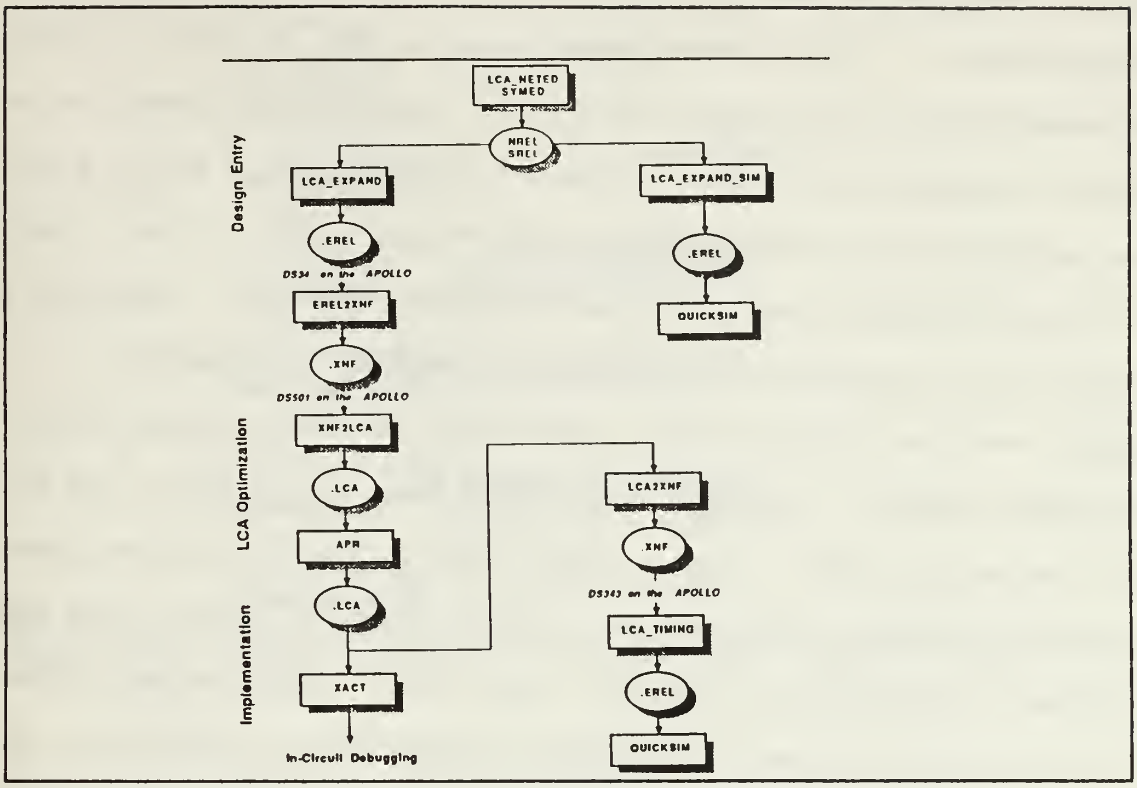
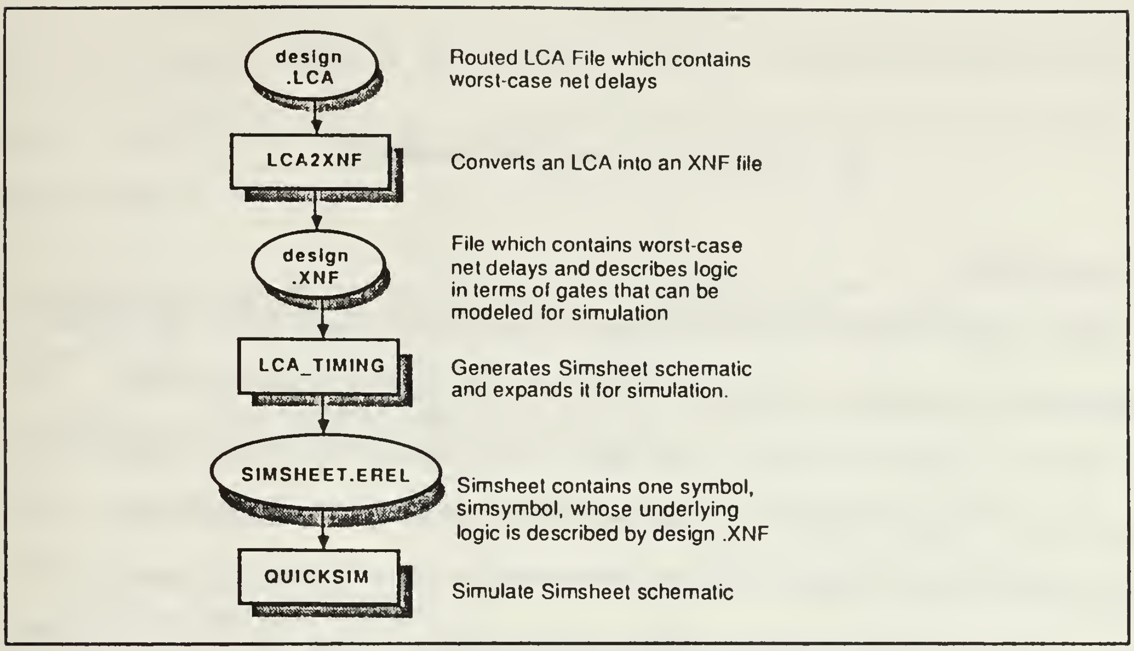


Figure 35 Apollo design flowpath [from Ref. 5]

use as additional building blocks if required. Once the design is laid out, it can be expanded and extracted using LCA_EXPAND_SIM for direct input into QUICKSIM logic simulation, or can be expanded using LCA_EXPAND for input into the design implementation software (see Figure 35). It is recommended that QUICKSIM be run on the design to verify the functional operations prior to taking it into the XILINX environment. To take the design across the interface boundary EREL2XNF is run which converts the ".erel" file from the MENTOR environment to a ".xnf" file in the XILINX environment. At this point it should be obvious to the designer that there is no design manager to guide one along the XILINX design path. Everything must be done manually from the command line environment. There is also an error in the XILINX documentation (see Figure 35) which implies that there is an XNF2LCA program like that in the PC environment. This is not the case and the conversion from the ".xnf" file to the ".lca" file must be done manually each step of the way. This will require at a minimum the running of XNFMAP and MAP2LCA. If there are any hierarchical drawings or PAL design files, PDS2XNF, XNFOPT, XNFMERGE and XNFDRC may also need to be run at the appropriate time. Once the ".lca" file has been created it can be placed and routed into an LCA by the APR program resulting in a routed ".lca" file. This routed ".lca" file can then be used in MAKEBITS and MAKEPROM to generate the configuration program data required to actually configure the LCA to the design. The next step will be to backannotate the routed ".lca" file and input it into the simulator for design verification. This is accomplished by running LAC2XNF to convert the routed ".lca" file to an ".xnf" file. LCA_TIMING is then run which provides a file "simsheet.erel" which can be input into QUICKSIM. This is an important step. This allows the simulation of the design once it is place and routed in the LCA. Figure 36 shows the backannotation process. The significance of this is that by comparing the pre-routing and post-routing simulation results actual design performance inside the LCA can be verified.


design .LCA
Routed LCA File which contains worst-case net delays
LCA2XNF
Converts an LCA into an XNF file
design .XNF
File which contains worst-case net delays and describes logic in terms of gates that can be modeled for simulation
LCA_TIMING
Generates Simsheet schematic and expands it for simulation.
SIMSHEET.EREL
Simsheet contains one symbol, simsymbol, whose underlying logic is described by design .XNF
QUICKSIM
Simulate Simsheet schematic
Figure 36 Backannotation of LCA design [from Ref. 5]

D. A COMPARISON OF DESIGN PLATFORMS

The engineering workstation as compared to the PC is a superior platform with regard to performance and degree of integration. The PC serves as a better design platform for the first time user. The MS-DOS file system is easier to understand and the design entry and design implementation interface are more accurately documented. The PC platform software is obviously more mature and the design manager which is totally menu-driven makes the design process a lot easier. There are some serious memory limitations in our PC environment. The Apollo workstation is approximately five times as fast as the IBM P/S 2. On large designs this would be a big advantage.

V. SCSI - A DESIGN EXAMPLE

A. OVERVIEW

The first step in doing any design is to break the design problem into small pieces that can be easily understood. In the SCSI standard there are many options available, and most available characteristics are implementation dependent. The implementation chosen for this thesis is a device capable of carrying out five basic SCSI functions. The device must be capable of detecting when the SCSI bus is free and begin arbitrating for the bus after the appropriate delay. The device must be capable of selecting a target. In the case when it is a target, it must be capable of acknowledging that it has been selected. The heart of the SCSI is an eight bit data bus that must be capable of transfering data in both directions. For simplicity, the assumption is made that this device has a SCSI ID of 4, that is, bit 4 is equal to 1. In order to simplify the design and to more easily understand all of the SCSI functions, five circuits were drawn seperately in both FUTURENET and MENTOR GRAPHICS environments. This allowed for timing simulation of the MENTOR GRAPHICS drawings with QUICKSIM to verify individual crcuit operation. The designs would then be implemented on a PC using the FUTURENET drawings via the FUTURENET/XILINX interface. Initial implementation was done using XC2064 parts but a final implementation was done using XC3020 parts. The five circuits involved in this design are:

- BUSFREE which detects when the SCSI bus is free and signals arbitration to begin after the appropriate time delay.
- ARBIT which arbitrates the SCSI bus to the device with the highest SCSI ID bit.
- SELECT which selects a target for a SCSI initiator.
- SELTAR which allows a SCSI device to acknowledge that it is a selected target.
- INFOTRAN which allows information to travel bi-directionaly to and from a SCSI device as well as not interfer with other SCSI devices on the bus.

It should be noted that for simplicity, only the data transfer will be considered. Message and Command transfers would only involve the target driving two control lines to the initiator.

Once the design implementation was completed on the PC for the five circuits, the design was redone in the engineering workstation environment. This time however, backannotation was performed on the implemented design to see the effects of LCA placement and routing on design performance speed.

B. INITIAL DESIGN LAYOUT

As previously discussed, the overall SCSI design was broken up into five functional circuits. A description of what these functions performed was specified and transformed into schematics. The schematics were laid out using Mentor

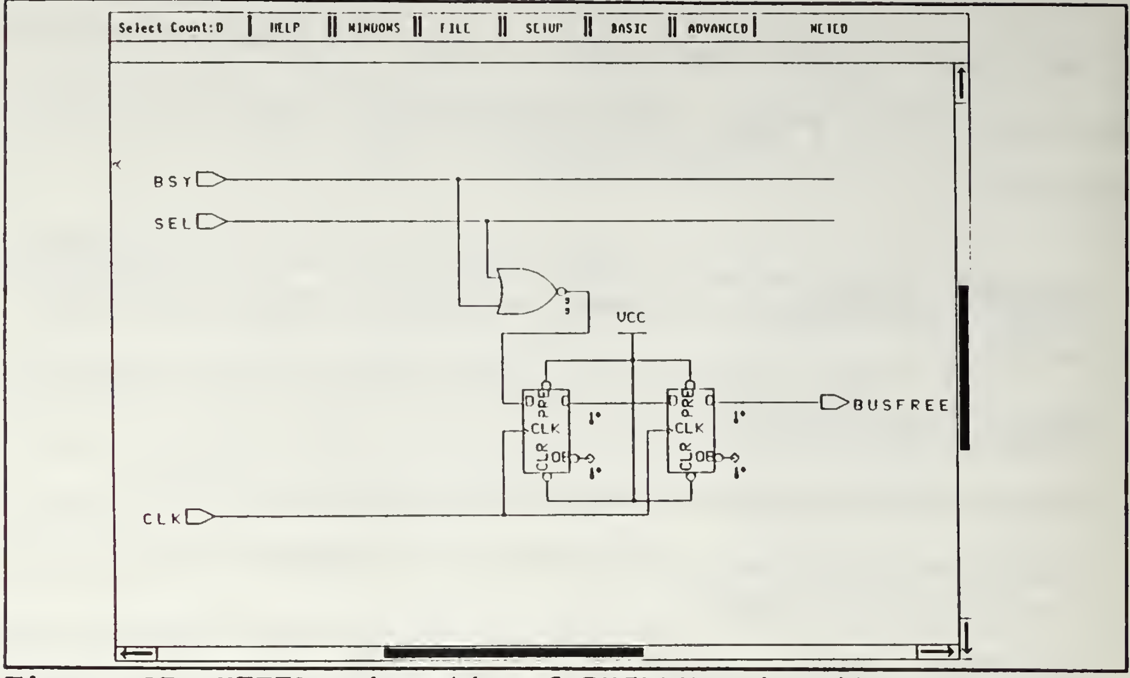
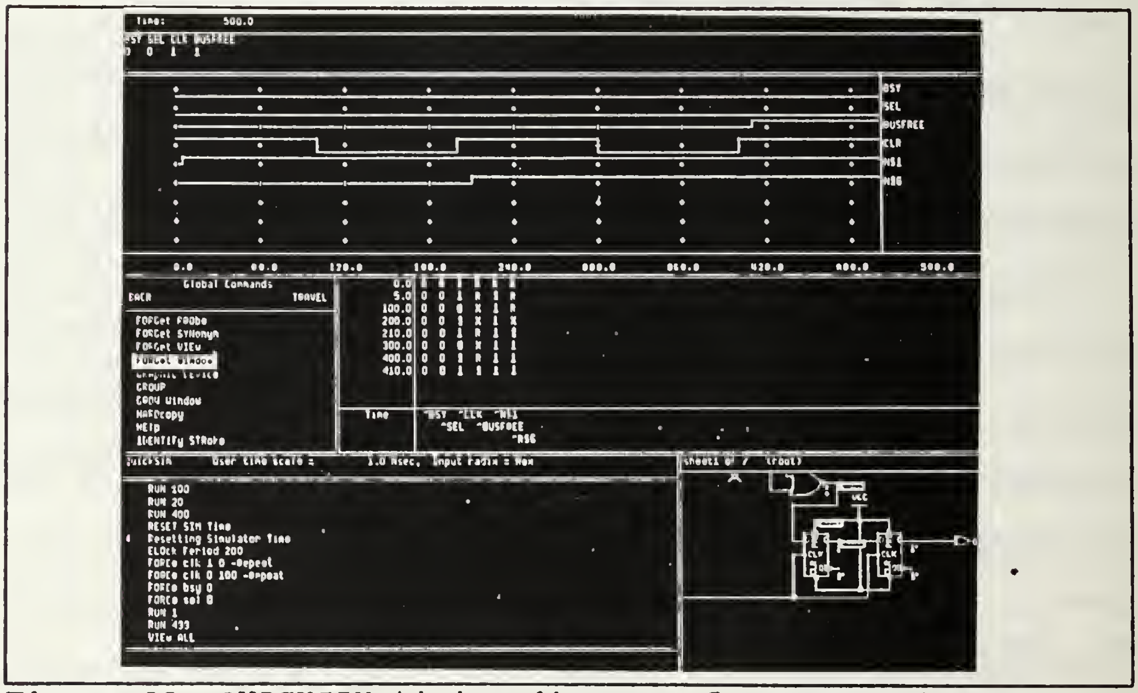


Figure 37 NETED schematic of BUSFREE circuit



```
     0.0
     5.0  0  0  1  R  1  R
   100.0  0  0  0  X  1  R
   200.0  0  0  1  X  1  X
   210.0  0  0  1  R  1  1
   300.0  0  0  0  X  1  1
   400.0  0  0  1  R  1  1
   410.0  0  0  1  1  1  1

    Time   ^BSY  ^CLK  ^NS1
             ^SEL  ^BUSFREE
                        ^NS6
```

```
QUICKSIM      User time scale =      1.0 Nsec,  Input radix = Hex

   RUN 100
   RUN 20
   RUN 400
   RESET SIM Time
   Resetting Simulator Time
   CLOck Period 200
   FORCe clk 1 0 -Repeat
   FORCe clk 0 100 -Repeat
   FORCe bsy 0
   FORCe sel 0
   RUN 1
   RUN 433
   VIEw ALL
```

Figure 38 QUICKSIM timing diagram of BUSFREE circuit

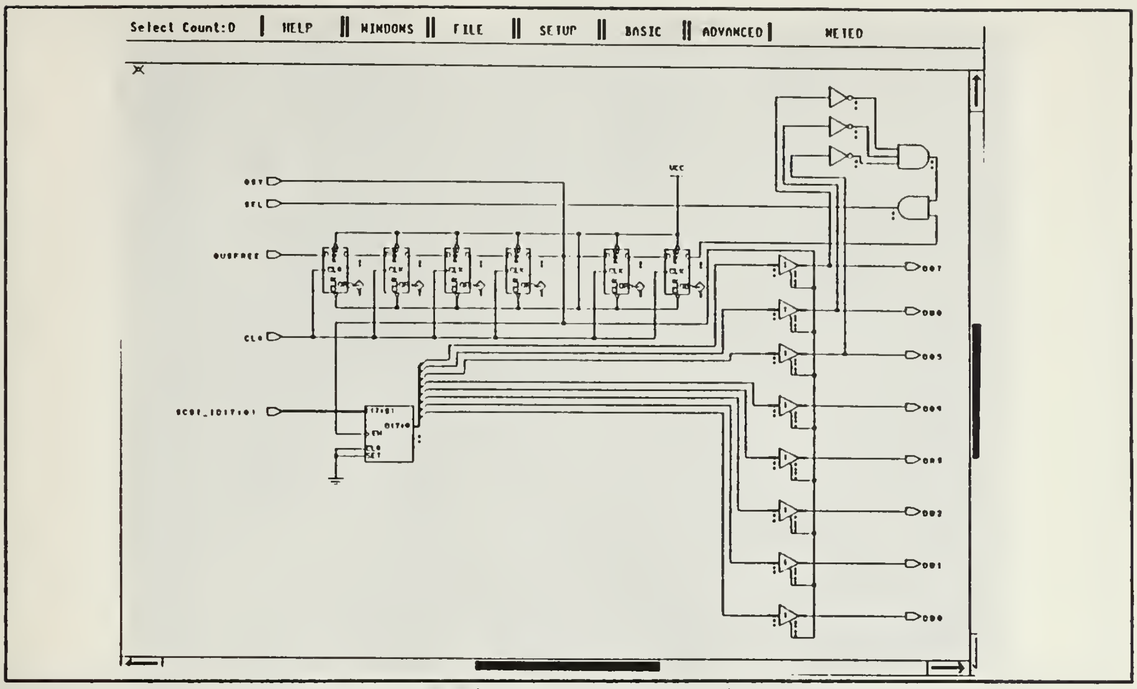
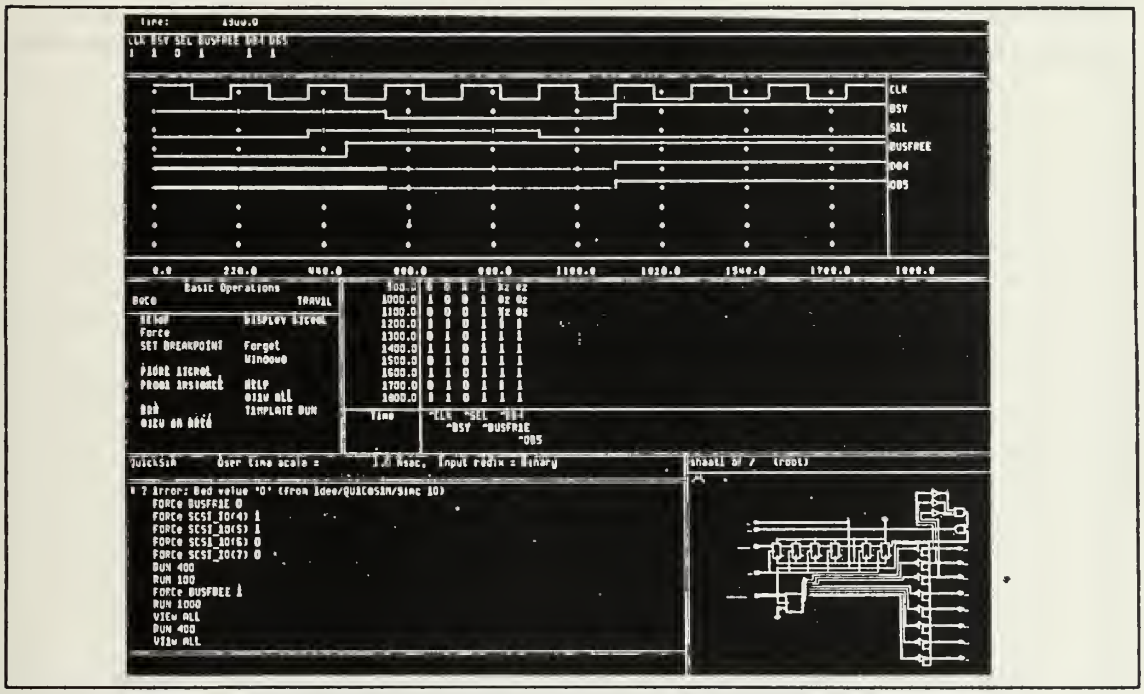


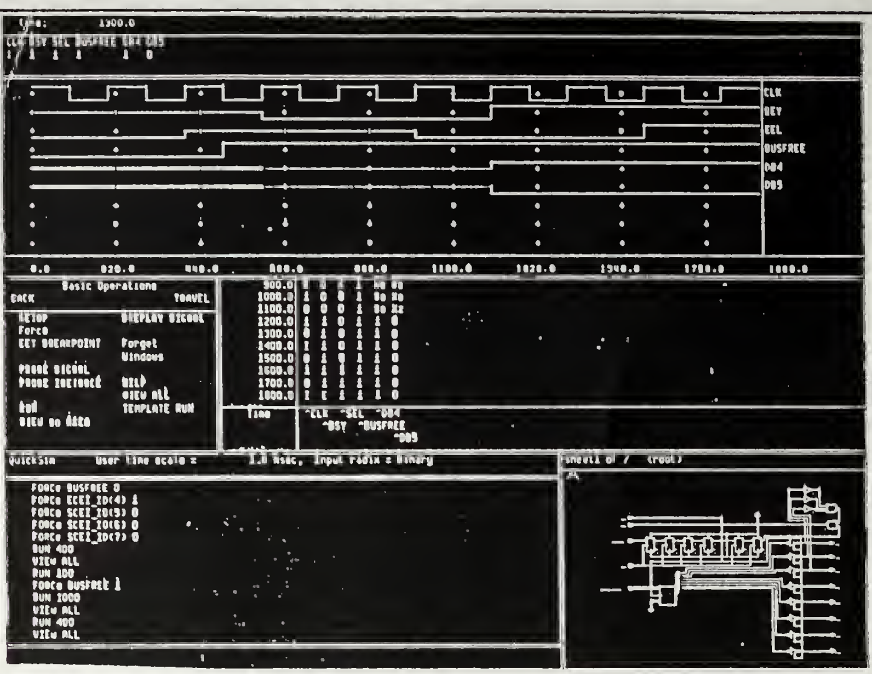
Figure 39 NETED schematic of ARBIT circuit



```
  900.0 | 0 0 0 1 Xz 0z
 1000.0 | 1 0 0 1 0z 0z
 1100.0 | 0 0 0 1 Xz 0z
 1200.0 | 1 1 0 1 1  1
 1300.0 | 0 1 0 1 1  1
 1400.0 | 1 1 0 1 1  1
 1500.0 | 0 1 0 1 1  1
 1600.0 | 1 1 0 1 1  1
 1700.0 | 0 1 0 1 1  1
 1800.0 | 1 1 0 1 1  1
 Time   | ^CLK ^SEL ^OD4
            ^BSY ^BUSFREE
                      ^OD5
```

```
QuickSim     User time scale =   1.0 Nsec,  Input radix = binary

? Error: Bad value 'O' (from [illegible]/QUICKSIM/[illegible] 10)
   FORCE BUSFREE 0
   FORCE SCSI_IO(4) 1
   FORCE SCSI_IO(5) 1
   FORCE SCSI_IO(6) 0
   FORCE SCSI_IO(7) 0
   RUN 400
   RUN 100
   FORCE BUSFREE 1
   RUN 1000
   VIEW ALL
   RUN 400
   VIEW ALL
```

Figure 40 QUICKSIM timing diagram of ARBIT circuit with arbitration lost



```
  Time    ^CLK ^SEL ^DB4
            ^BSY ^BUSFREE
                       ^DB5
 900.0
1000.0   1 0 0 1 X0 X0
1100.0   0 0 0 1 X0 X2
1200.0   1 1 0 1 1 0
1300.0   0 1 0 1 1 0
1400.0   1 1 0 1 1 0
1500.0   0 1 0 1 1 0
1600.0   1 1 1 1 1 0
1700.0   0 1 1 1 1 0
1800.0   1 1 1 1 1 0
```

```
QuickSim     User time scale =     1.0 Nsec, Input radix = Binary

FORCE BUSFREE 0
FORCE SCEI_ID(4) 1
FORCE SCEI_ID(5) 0
FORCE SCEI_ID(6) 0
FORCE SCEI_ID(7) 0
RUN 400
VIEW ALL
RUN 100
FORCE BUSFREE 1
RUN 1000
VIEW ALL
RUN 400
VIEW ALL
```

Figure 41 QUICKSIM timing simulation of ARBIT with arbitration won

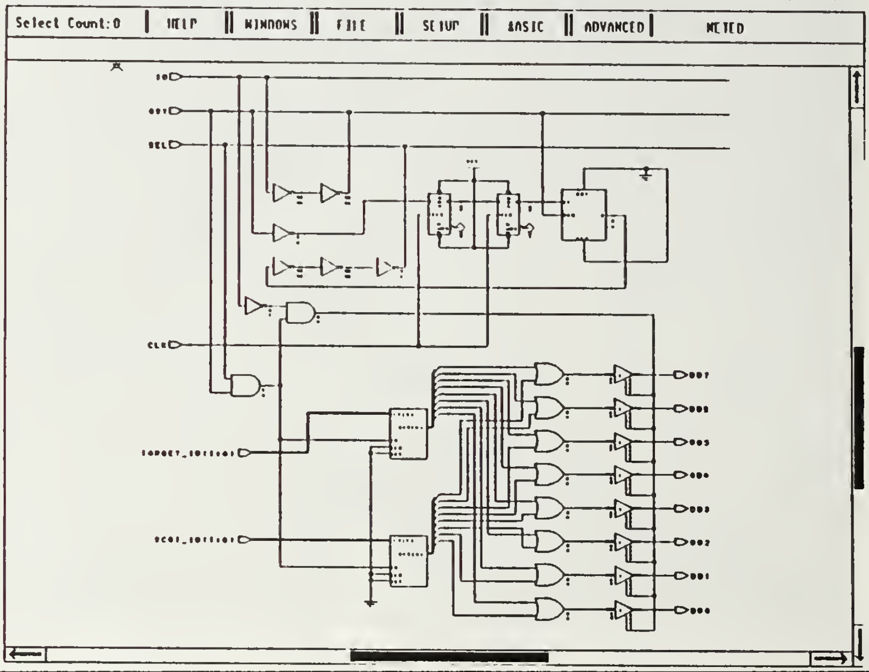
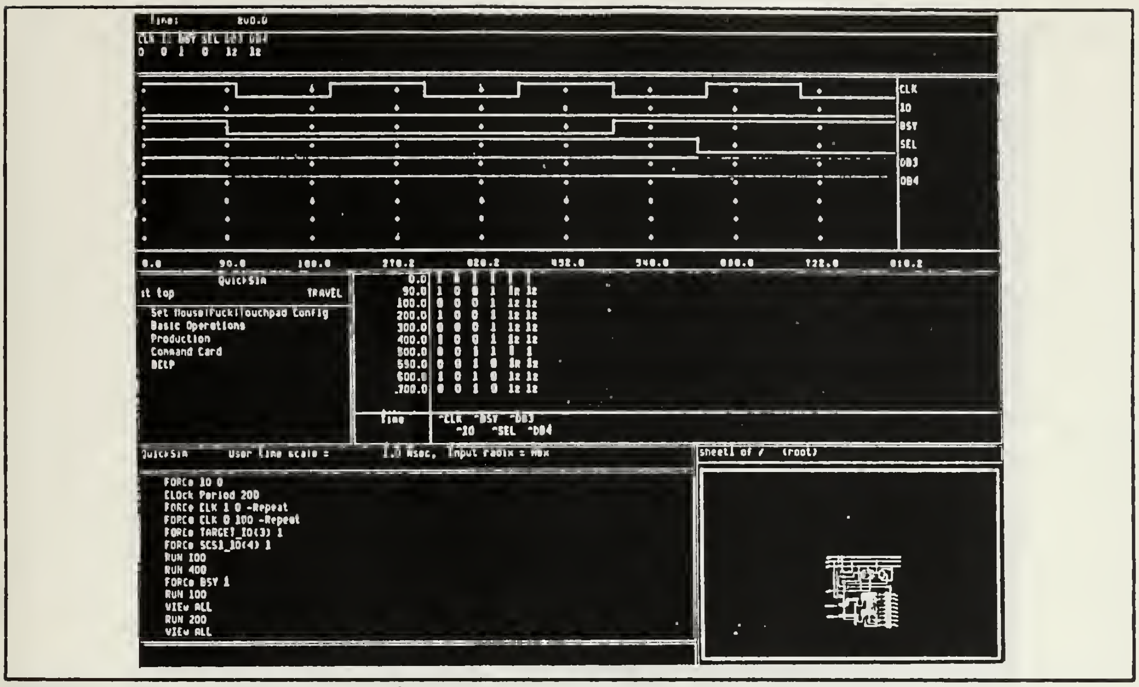


Figure 42 NETED schematic of SELECT circuit



```
Time:          800.0
CLK IO BSY SEL OB3 OB4
0   0  1   0   1z  1z

0.0    90.0    180.0    270.2    020.2    452.0    540.0    080.0    722.0    810.2

            QuickSim
st top                  TRAVEL
  Set Mouse|Puck|Touchpad Config
  Basic Operations
  Production
  Command Card
  HELP

   0.0
  90.0  1  0  0  1  1z 1z
 100.0  0  0  0  1  1z 1z
 200.0  1  0  0  1  1z 1z
 300.0  0  0  0  1  1z 1z
 400.0  1  0  0  1  1z 1z
 500.0  0  0  1  1  1  1
 550.0  0  0  1  0  1z 1z
 600.0  1  0  1  0  1z 1z
 700.0  0  0  1  0  1z 1z
  Time  ^CLK  ^BSY  ^OB3
           ^IO   ^SEL  ^OB4

QuickSim      User time scale =       1.0 nsec,  Input radix = Hex
    FORCe IO 0
    CLOck Period 200
    FORCe CLK 1 0 -Repeat
    FORCe CLK 0 100 -Repeat
    FORCe TARGET_IO(3) 1
    FORCe SCSI_IO(4) 1
    RUN 100
    RUN 400
    FORCe BSY 1
    RUN 100
    VIEw ALL
    RUN 200
    VIEw ALL

sheet1 of /   (root)
```

Figure 43 QUICKSIM timing simulation of SELECT circuit

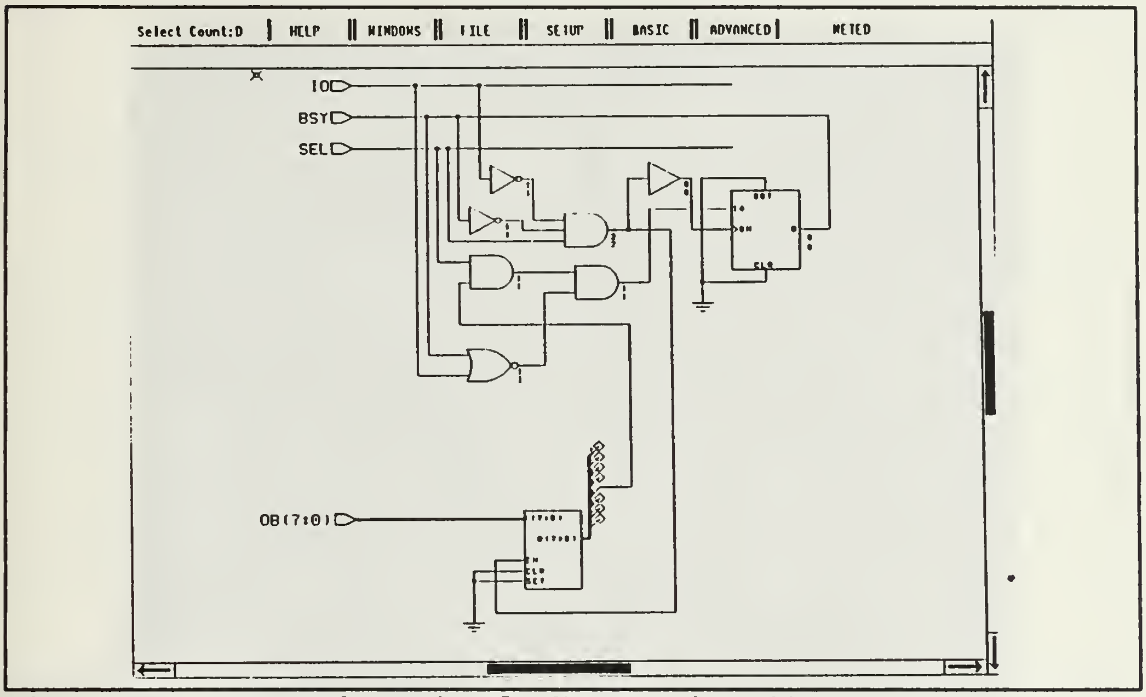
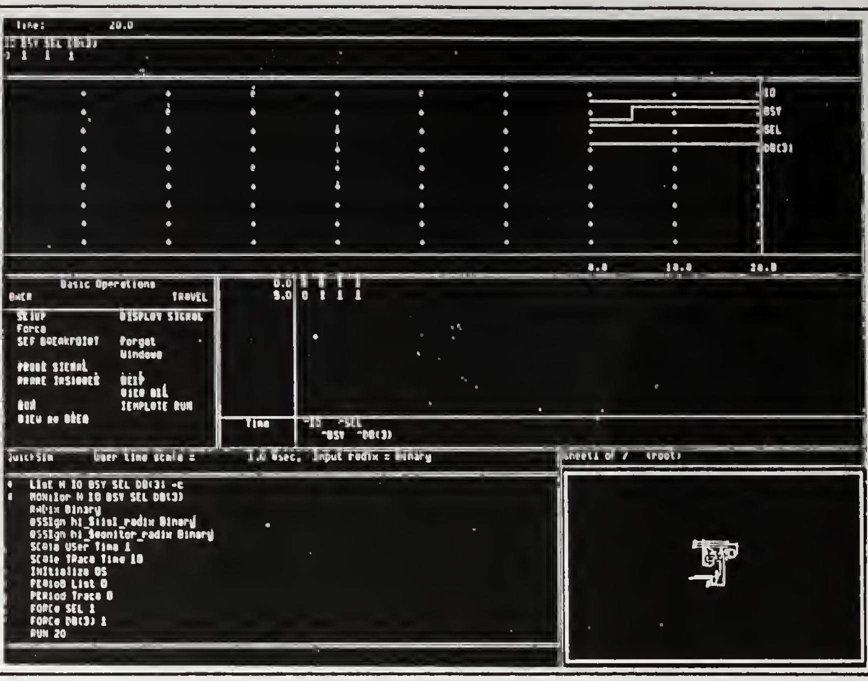
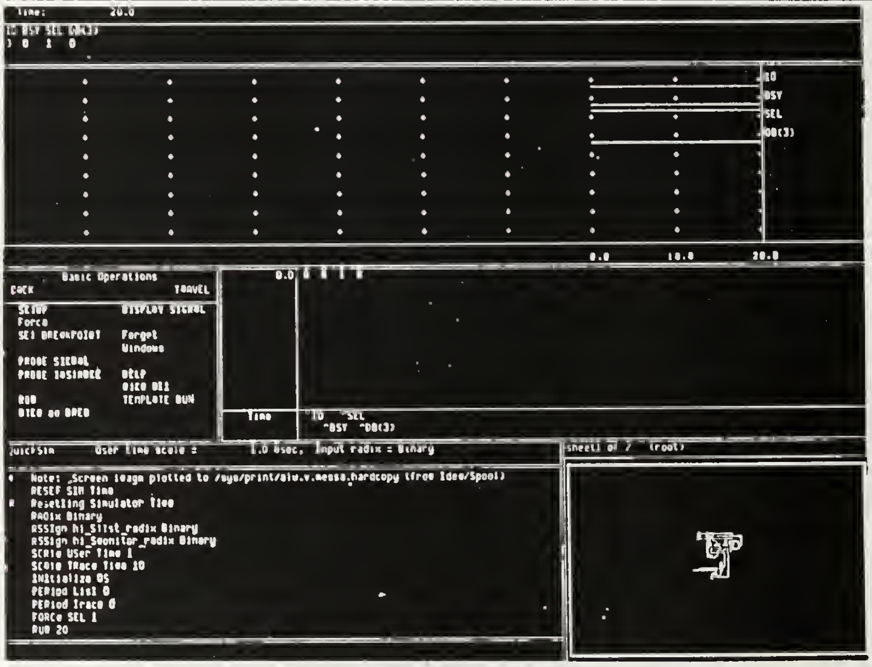


Figure 44 NETED layout of SELTAR circuit



```
QuickSim        User time scale =        1.0 usec,  Input radix = Binary
   LIst H IO BSY SEL DB(3) -c
   MONitor H IO BSY SEL DB(3)
   RAdix Binary
   ASSIgn hi_Slist_radix Binary
   ASSIgn hi_Smonitor_radix Binary
   SCale USer Time 1
   SCale TRace Time 10
   INItialize 0S
   PERiod List 0
   PERiod Trace 0
   FORCe SEL 1
   FORCe DB(3) 1
   RUN 20
```

Figure 45 QUICKSIM timing of SELTAR circuit when selected



```
QuickSim        User time scale =        1.0 usec,  Input radix = Binary
   Note: Screen image plotted to /sys/print/alw.v.messa.hardcopy (from Idea/Spool)
   RESET SIM Time
   Resetting Simulator Time
   RAdix Binary
   ASSIgn hi_Slist_radix Binary
   ASSIgn hi_Smonitor_radix Binary
   SCale USer Time 1
   SCale TRace Time 10
   INItialize 0S
   PERiod List 0
   PERiod Trace 0
   FORCe SEL 1
   RUN 20
```

Figure 46 QUICKSIM timing of SELTAR circuit when not selected

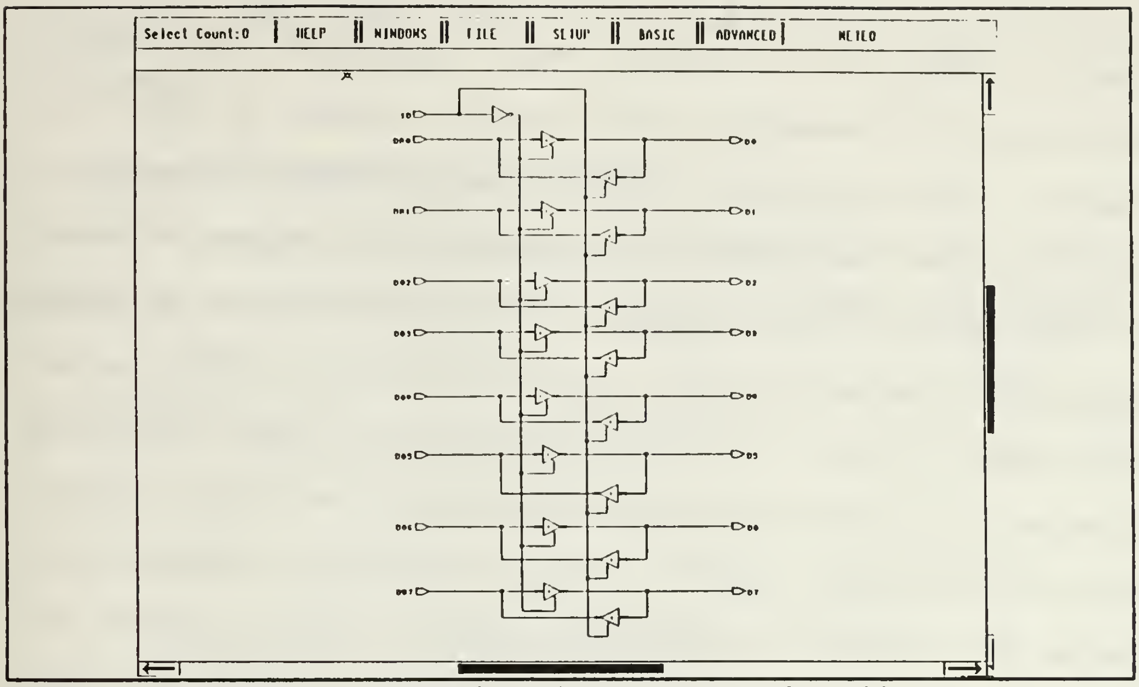
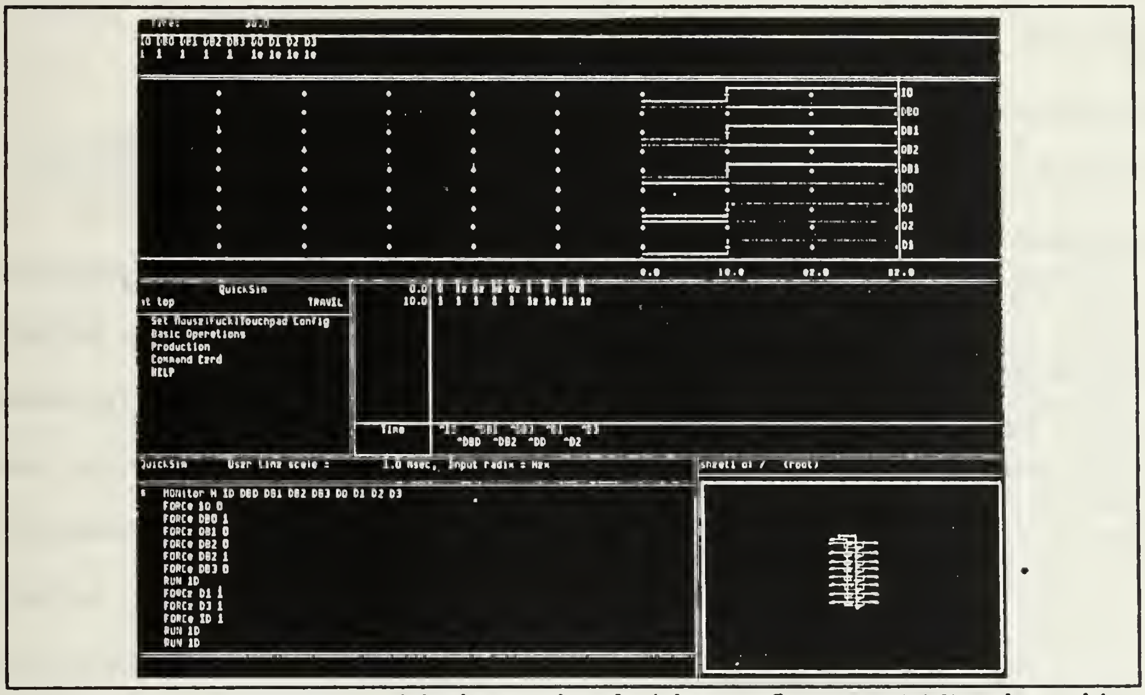


Figure 47 NETED schematic of INFOTRAN circuit



```
QuickSim      User Line scale =      1.0 nsec,  Input radix = Hex
$   MONitor H ID DB0 DB1 DB2 DB3 D0 D1 D2 D3
    FORCe ID 0
    FORCe DB0 1
    FORCe DB1 0
    FORCe DB2 0
    FORCe DB2 1
    FORCe DB3 0
    RUN 10
    FORCe D1 1
    FORCe D3 1
    FORCe ID 1
    RUN 10
    RUN 10
```

Figure 48 QUICKSIM timing simulation of INFOTRAN circuit

Graphics NETED and tested using QUICKSIM to verify functionality of operation and initial timing of the five circuits. More accurate and meaningful timing can only be simulated after the design is backannotated. Figure 37 shows the NETED layout of the BUSFREE circuit while Figure 38 shows the QUICKSIM timing results. Figure 39 shows the NETED layout of the ARBIT circuit while Figures 40 and 41 show its timing characteristics with arbitration both won and lost. Figures 42 and 43 show the SELECT circuit layout and timing while Figures 44, 45 and 46 show the SELTAR circuit layout and timing. Finally, Figures 47 and 48 show the main SCSI bus INFOTRAN layout and timing. A detailed analysis of timing performance will be examined later in this chapter. This initial layout was done using components from the gen_lib components library within NETED. This posed a problem in later development stages since the gen_lib components possessed slightly different characteristics than the XILINX macro library or FUTURENET components and as a result some design changes were required. A good example of this was the tri-state buffer which was a crucial component in the design. In the Mentor Graphics environment, a low enabling signal caused the device to go to the high impedance output condition while the XILINX macro three-state buffers operated in the opposite manner with a high enabling signal causing the device to go to the high impedance output condition. Once the designs were laid out and were operating satisfactorily in simulation, they were taken for implementation into the PC environment.

C. DESIGN IMPLEMENTATION ON THE PC

The implementation process involved taking the circuit schematics and redrawing them in the FUTURENET environment.

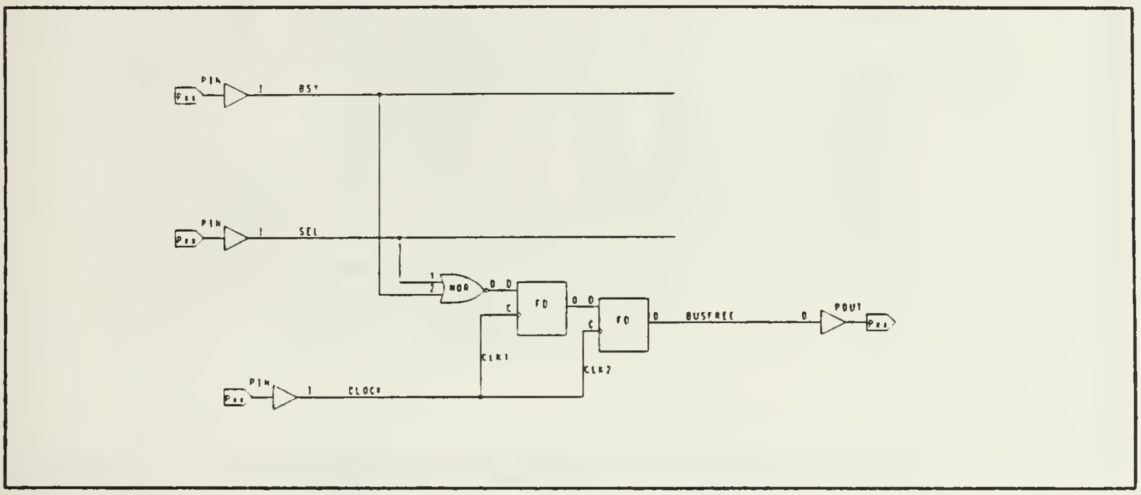


Figure 49 FUTURENET schematic of BUSFREE circuit

Figures 49, 50, 51, 52 and 53 show the BUSFREE, ARBIT, SELECT, SELTAR and INFOTRAN circuits as drawn with FUTURENET. These designs were originally done using XC2000 family library. When they were converted to XC3000 family designs one item that had to be changed was the addition of the global clock buffer (GCLK). This was forced by the software. If an external clock was being used to toggle more than two flip-flops and GCLK was not being used, the software would flag a design error and identify the nets effected. It is recommended that the net with the highest fan-out be the net to use the GCLK resource.

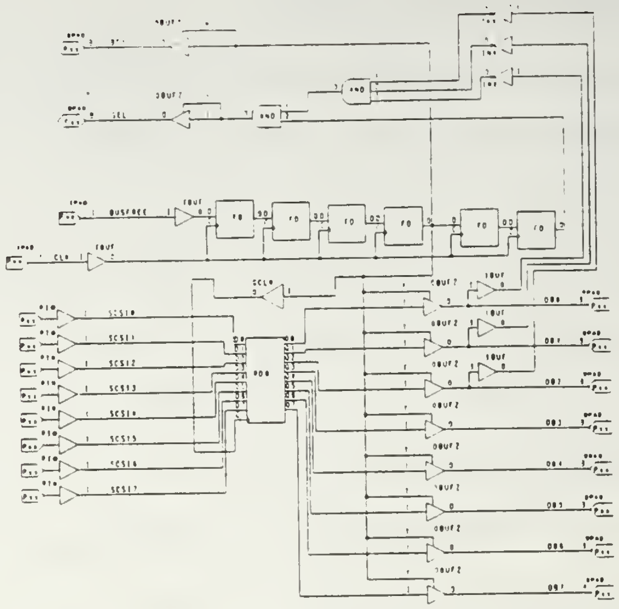


Figure 50 FUTURENET schematic of ARBIT circuit

On the PC, the capability to functionally simulate design performance was not available. As a result, once implementation was completed, there was no way to determine what effects routing and placement into the LCA had on design performance. The initial design implementation was done using XC2064 LCAs which had the capacity of 1200 gate designs. After upgrading system the memory to handle larger designs, the designs were reimplemented using XC3020 LCAs. As previously mentioned, at this point a design error was identified. The XC3000 family LCAs have a special global

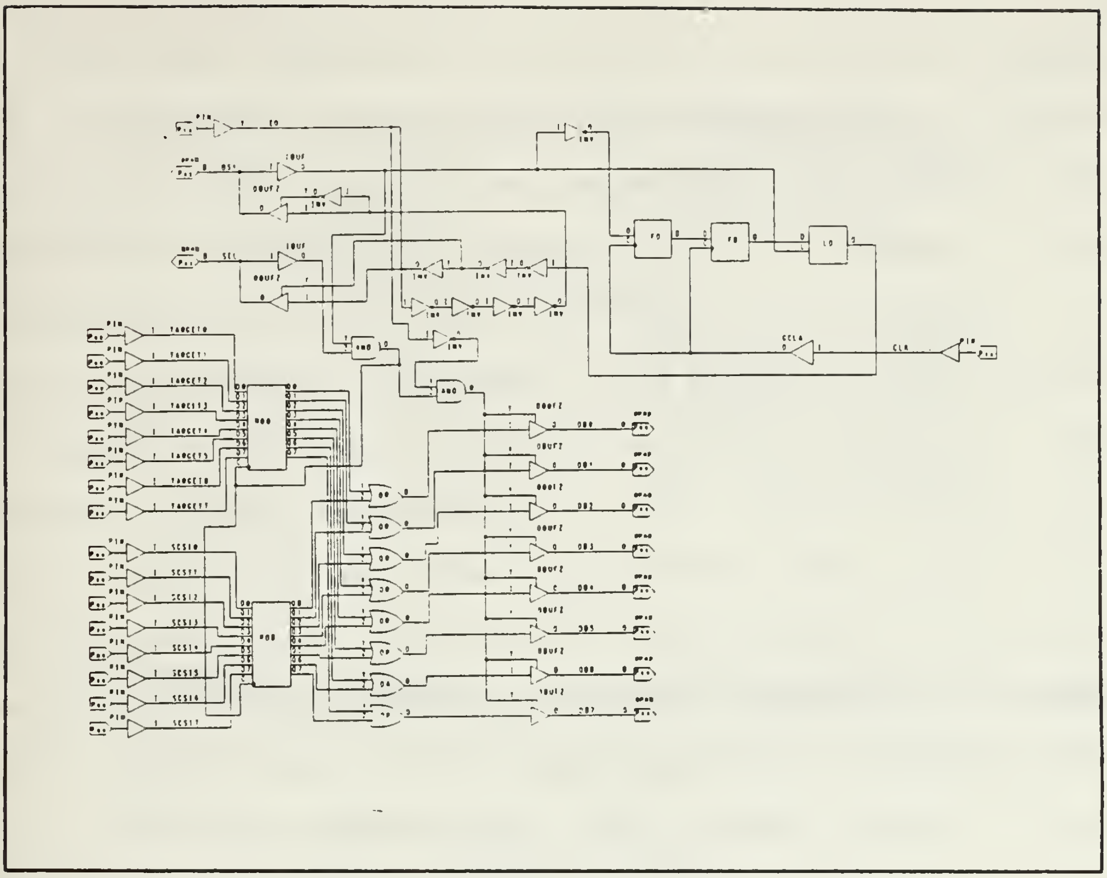


Figure 51 FUTURENET schematic of SELECT circuit

clock buffer (GCLK) which is used to minimize the effects of fan-out on clock signals caused by trying to drive many flip-flops. An alternate clock buffer (ACLK) is also available for the same purpose. The software forces the designer to use these resources. It is recommended that the sources with the highest fan-out be given the GCLK and ACLK resources.

Implementation on the PC platform is quite simple. It starts with running the XILINX Design Manager (XDM) which is essentially a design environment shell capable of calling all design tools from one environment. XDM is initially run from DOS and is setup in accordance with a user specified profile.

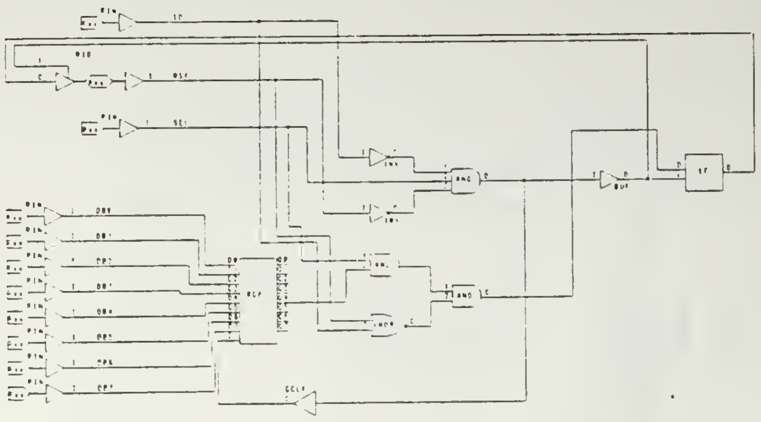


Figure 52 FUTURENET schematic of SELTAR circuit

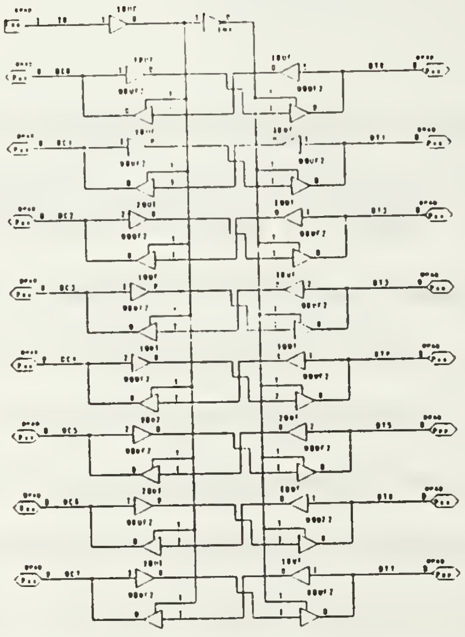


Figure 53 FUTURENET schematic of INFOTRAN circuit

The first menu DesignEntry is used to call an entry program (i.e., FUTURENET) and the associated library files. Translation of the SCSI design files is performed automatically by XMAKE which takes the design file, for example, from "busfree.dwg" which is a FUTURENET drawing to "busfree.lca" which is an unrouted ".lca" file. The PlaceRoute menu allows the running of APR which takes the unrouted "busfree.lca" and creates a routed "busfree.lca". The choice was made to utilize an EPROM to hold the configuration data for the LCA. This is accomplished by running the XACT program. Initially, memory constraints prohibited the running of XACT from the XDM shell. The PC used in this thesis has been upgraded to 4 MB which alleviates this problem. The primary purpose of the XACT program is to manually edit LCA routing, but it also acts as the shell for two other important programs which are Makebits and Makeprom. In this example, the routed "busfree.lca" file is converted to a configuration bitstream file "busfree.bit" by the Makebits program. This bitstream was then converted to a EPROM programming file "busfree.mcs" which is in Intel Hex format (MCS-86). The next thing to accomplish was actually programming the EPROM. Two methods were used to accomplish this. An expensive DATA I/O universal programmer was used. It could however only work on certain type of EPROMs since it did not support some of the newer CMOS technology EPROMs (e.g., AM27C64). The second method involved the use of an inexpensive PC based EPROM programmer. The strategy in this method was to convert the EPROM programming file to pure binary code and to download it to the EPROM directly. This was accomplished with a program HEX2BIN which was provided with the programmer. One final issue that was looked at with regard to EPROM programming was the starting address of the configuration program. The MakeProm program allow the designer to specify the address at which the configuration program should reside. For example, 1853 bytes are required for an XC3020 LCA. Thus, an 8 KB EPROM could hold four seperate configuration programs. It should be noted that the size of the configuration program is a function of the actual LCA device used in the design and not the design itself. Both EPROM programming methods used allowed this feature to be exploited. They both allowed the configuration program to be stored at any address in the EPROM. This design example was not overly complex so that this feature was not utilized. However, it could be very useful in a more complex design.

D. DESIGN IMPLEMENTATION ON AN ENGINEERING WORKSTATION

For this part of the work, design was implemented utilizing an Apollo 4000 series workstation with Mentor Graphics tools and the XILINX Development System software. Aditionally, an interface module was also installed which provided the bridge between the MENTOR tools and the LCA design implementation. The design process involved the following design methodology:

1. Using LCA_NETED which uses XILINX macro libraries, capture schematic.

2. Using LCA_EXPAND_SIM, prepare the design file for QUICKSIM simulation.

3. Run QUICKSIM and thoroughly test the design.

4. Run LCA_EXPAND to EXPAND the design file in preparation for leaving the Mentor Graphics environment and entering the XILINX environment.

5. Run EREL2XNF to convert the Mentor Graphics design file to a XILINX Netlist File (XNF). A specific LCA part number must be provided at this point.

6. Run XNFMAP and MAP2LCA to convert the .xnf file to an unrouted .lca file.

7. Run APR to place and route the design.

8. Run LCA2XNF on the routed design which will provide an XNF file with routing delay information.

9. Run LCA_TIMING on the backannotated XNF file which prepares a new SIMSHEET for use in QUICKSIM.

10. Run QUICKSIM to verify design performance of the routed design.

11. Once design performance is verified, run MAKEBITS and MAKEPROM on the design file to get the configuration bitstream and EPROM programming file.

In this design example, this methodology was followed for each part of the SCSI design. The exception being, when generating the configuration bitstream, the EPROM programming file was only done for the completed design. In doing the design in this environment, several design errors were found as a result of being able to test the designs after routing. These errors were mainly caused by the fact that initially running QUICKSIM on a design file resulted in zero delays. However, the delays that were really there as a result of placement and routing in the LCA often caused signal conflicts. By breaking up the SCSI design into five functional blocks and thoroughly testing them in QUICKSIM both before and after routing, a much clearer view of the SCSI dsign characteristics was obtained. Also, by looking at smaller blocks of the design, a better understanding of LCA performance for design implementation with regard to routing delay effects was aquired.

The first part of the design that was attempted was also the most simple. The BUSFREE circuit merely was to detect when the SCSI bus was free and signal the beginning of

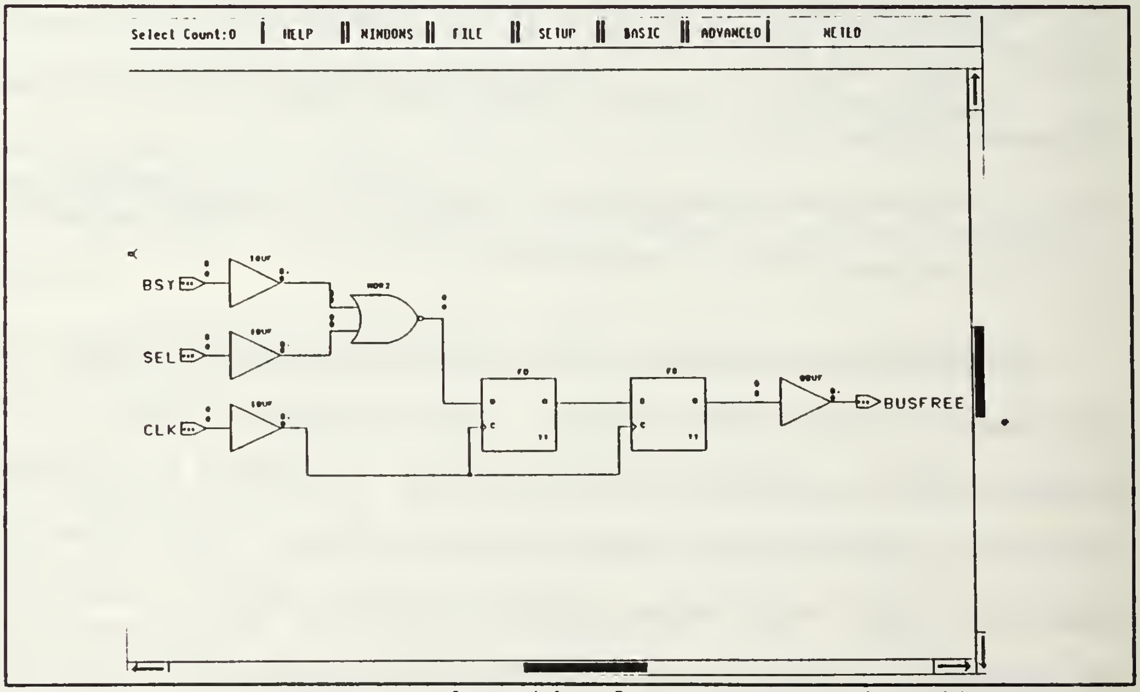
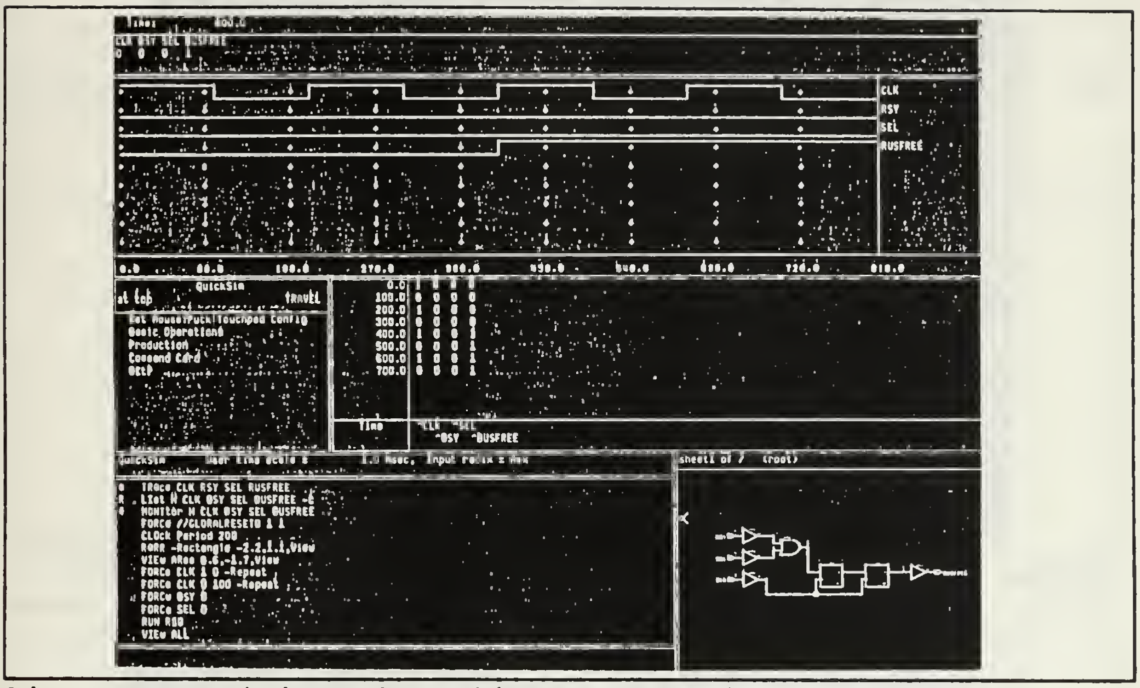


Figure 54 LCA_NETED schematic for BUSFREE circuit



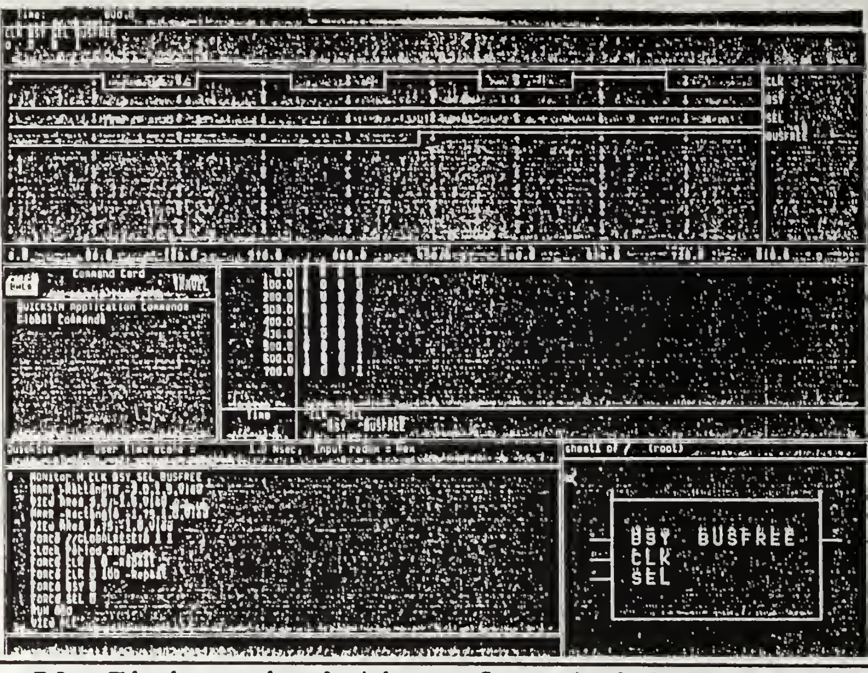
```
Time:         400.0
CLK BSY SEL BUSFREE
0 0 0 1
```

```
  0.0  1 0 0 0
100.0  0 0 0 0
200.0  1 0 0 0
300.0  0 0 0 0
400.0  1 0 0 1
500.0  0 0 0 1
600.0  1 0 0 1
700.0  0 0 0 1
 Time  ^CLK  ^SEL
          ^BSY  ^BUSFREE
```

```
QuickSim      User time scale is    1.0 Nsec.  Input radix = Hex
    TRAce CLK BSY SEL BUSFREE
    LISt H CLK BSY SEL BUSFREE -E
    MONItor H CLK BSY SEL BUSFREE
    FORCe //GLOBALRESETB 1 1
    CLOck Period 200
    RORR -Rectangle -2.2,1.1,View
    VIEw ARea 0.6,-1.7,View
    FORCe CLK 1 0 -Repeat
    FORCe CLK 0 100 -Repeat
    FORCe BSY 0
    FORCe SEL 0
    RUN 800
    VIEw ALL
```

Figure 55 Timing simulation of unrouted BUSFREE circuit

arbitration after 400 ns. Figure 54 shows the BUSFREE circuit. Figure 55 shows the timing for the unrouted design which shows the BUSFREE signal going high 400 ns after the BSY and SEL signals are both low. Figure 56 shows the timing simulation of the design that has been backannotated from the routed design which shows the BUSFREE signal going high 436 ns after BSY and SEL are both low. The extra 36 ns is a result of the routing but will not have much effect on performance since this design will just take a little longer than specified to begin the information transfer phase. Figure 57 shows the schematic for the ARBIT circuit which puts the SCSI ID on the SCSI bus at the beginning of the arbitration phase.



```
Monitor -M CLK BSY SEL BUSFREE
[illegible]
Force //GLOBALRESET [illegible]
Clock Period 200
Force CLK [illegible] -Repeat
Force CLK 0 100 -Repeat
Force BSY 0
Force SEL 0
Run [illegible]
```

Figure 56 Timing simulation of routed BUSFREE circuit

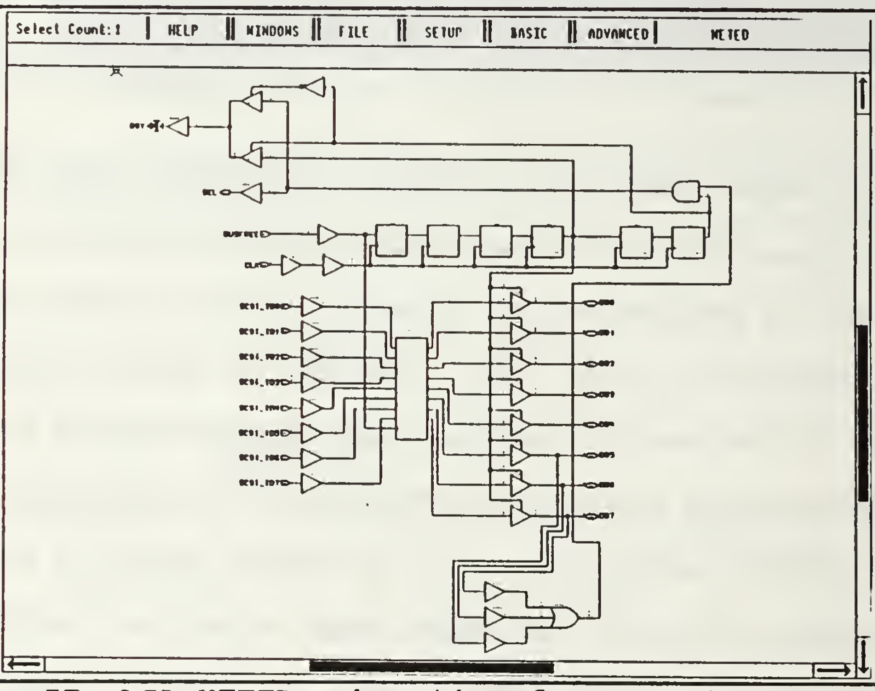
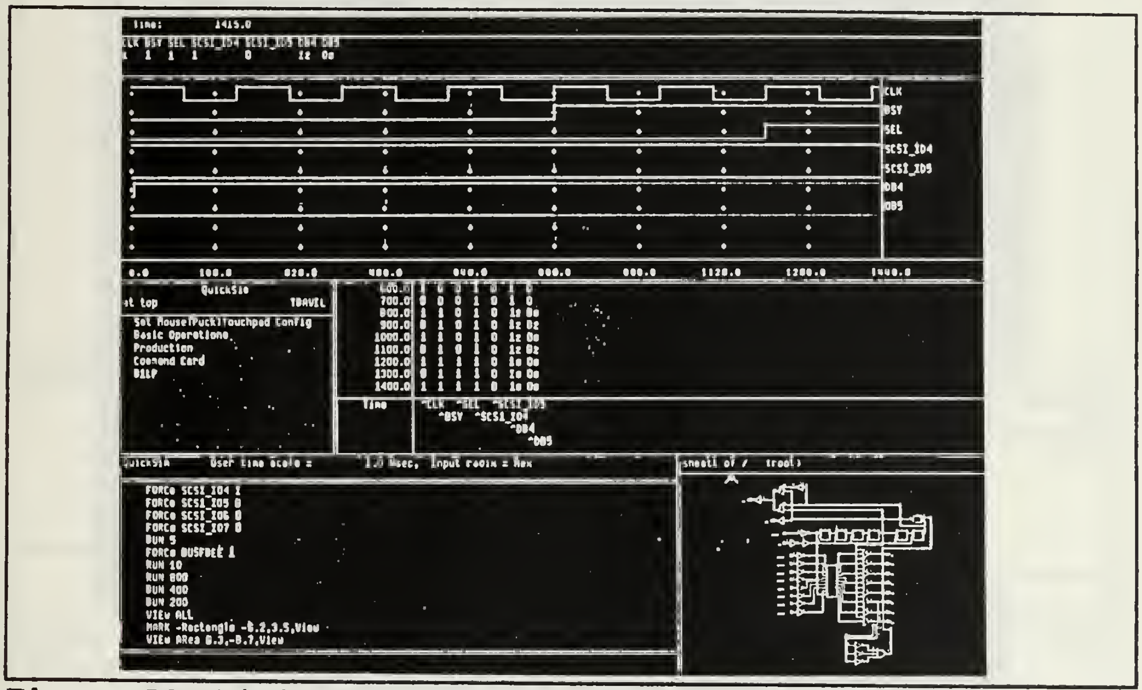


Figure 57 LCA_NETED schematic of ARBIT circuit

After 800 ns BSY is asserted and after 1200 ns SEL is asserted provided the arbitration was won. If arbitration was lost, BSY would be negated in lieu of asserting SEL. Due to a misinterpretation of the SCSI standard, the 1200 ns SEL



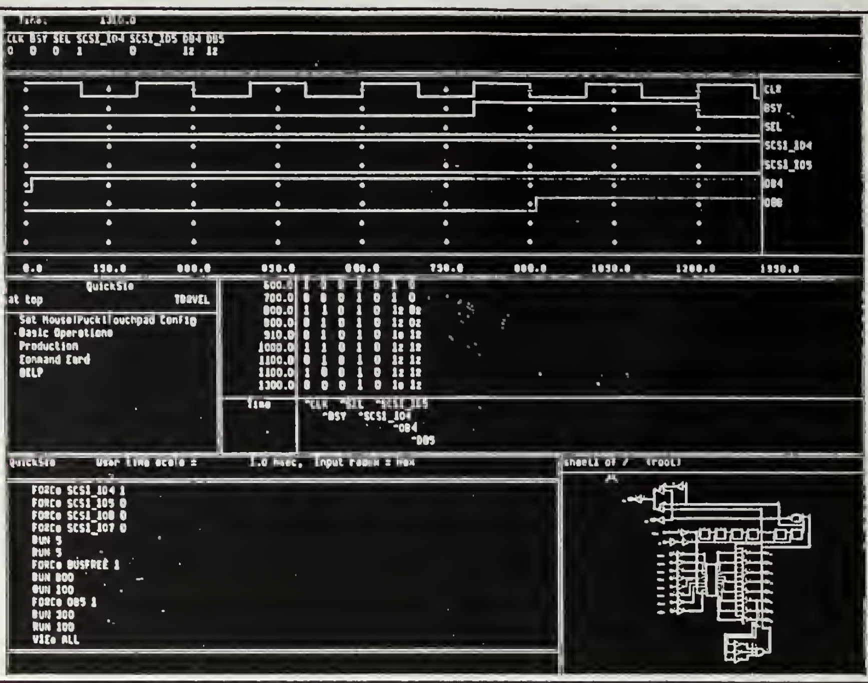
```
 400.0  1  0  0  1  0  1  0
 700.0  0  0  0  1  0  1  0
 800.0  1  1  0  1  0  1z 0e
 900.0  0  1  0  1  0  1z 02
1000.0  1  1  0  1  0  1z 0e
1100.0  0  1  0  1  0  1z 02
1200.0  1  1  1  1  0  1e 0e
1300.0  0  1  1  1  0  1e 0e
1400.0  1  1  1  1  0  1e 0e
  Time  ^CLK  ^SEL  ^SCSI_ID5
          ^BSY  ^SCSI_ID4
                  ^DB4
                     ^DB5
```

```
QuickSim     User time scale =     1.0 Nsec,  Input radix = Hex
FORCe SCSI_ID4 1
FORCe SCSI_ID5 0
FORCe SCSI_ID6 0
FORCe SCSI_ID7 0
RUN 5
FORCe BUSFREE 1
RUN 10
RUN 800
RUN 400
RUN 200
VIEw ALL
MARK -Rectangle -8.2,3.5,View
VIEw ARea 0.3,-8.7,View
```

Figure 58 Timing simulation of unrouted ARBIT circuit with arbitration won

assertion should be 1200 ns from the time that BSY is asserted making the SEL assertion 2000 ns from the time the arbitration phase begins. This error was corrected in the final design by adding four flip-flops clocked at 200 ns periods which increased the arbitration time by 800 ns consistent with the SCSI standard.

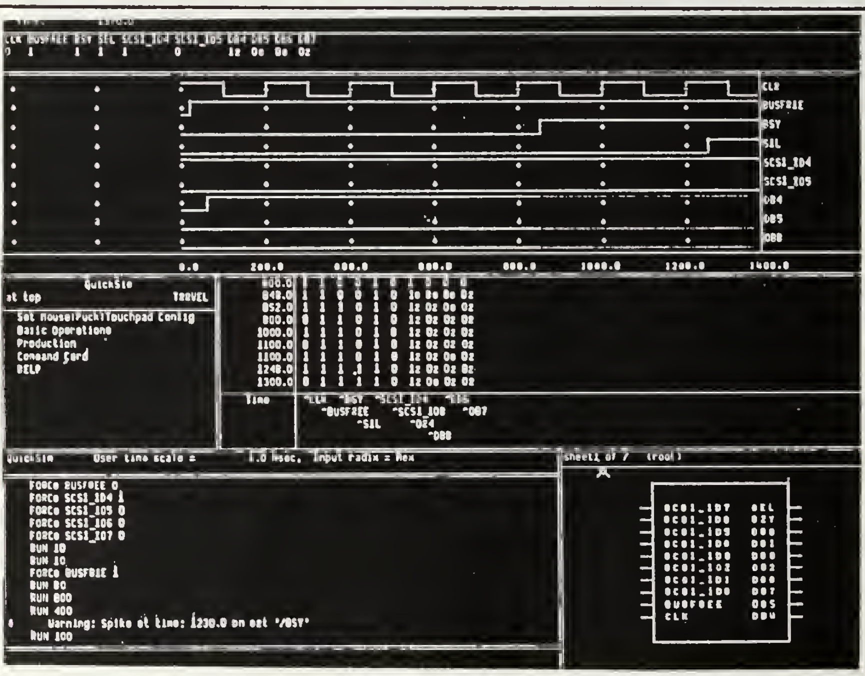
Figure 58 and 59 show the timing simulations of the unrouted ARBIT circuit when arbitration is won and lost,



```
 600.0 | 1 0 0 1 0 1 0
 700.0 | 0 0 0 1 0 1 0
 800.0 | 1 1 0 1 0 1z 0z
 800.0 | 0 1 0 1 0 1z 0z
 910.0 | 0 1 0 1 0 1o 1z
1000.0 | 1 1 0 1 0 1z 1z
1100.0 | 0 1 0 1 0 1z 1z
1100.0 | 1 0 0 1 0 1z 1z
1300.0 | 0 0 0 1 0 1o 1z
```

```
QuickSim      User time scale =      1.0 nsec,  Input radix = Hex

FORCE SCSI_ID4 1
FORCE SCSI_ID5 0
FORCE SCSI_ID0 0
FORCE SCSI_ID7 0
RUN 5
RUN 5
FORCE BUSFREE 1
RUN 800
RUN 100
FORCE DB5 1
RUN 300
RUN 100
VIEW ALL
```

Figure 59 Timing simulation for unrouted ARBIT circuit with arbitration lost

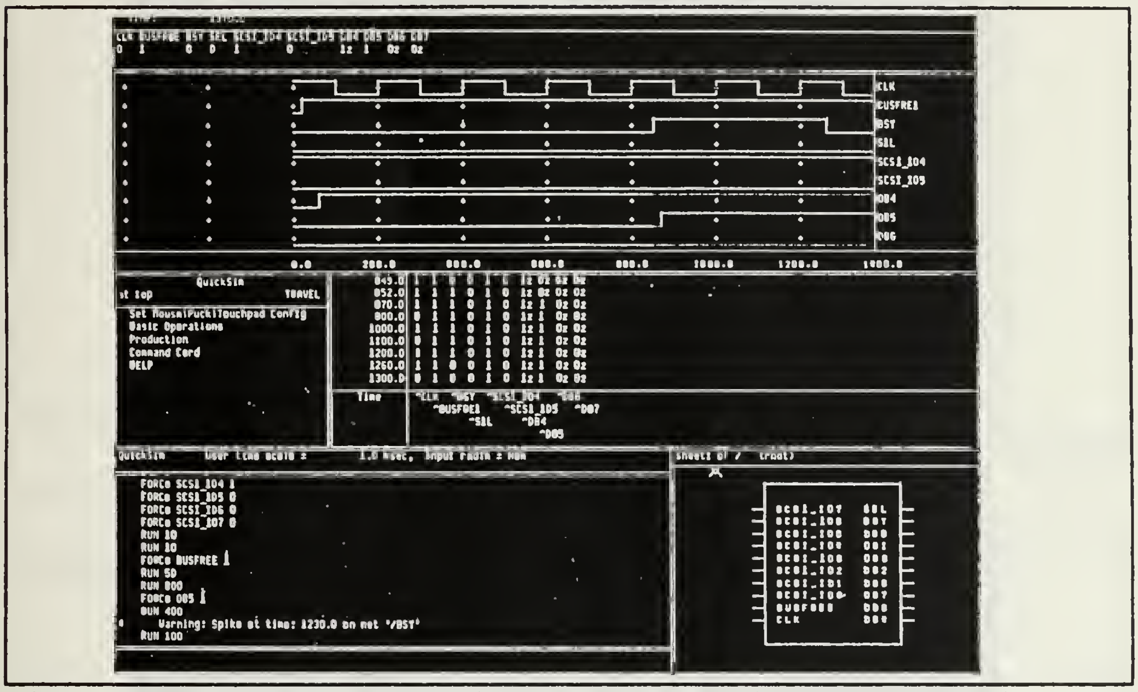


```
 800.0 | 1 1 0 0 1 0 1z 0z 0z 0z
 848.0 | 1 1 0 0 1 0 1o 0o 0o 0z
 852.0 | 1 1 1 0 1 0 1z 0z 0o 0z
 800.0 | 0 1 1 0 1 0 1z 0z 0z 0z
1000.0 | 1 1 1 0 1 0 1z 0z 0z 0z
1100.0 | 0 1 1 0 1 0 1z 0z 0z 0z
1100.0 | 1 1 1 0 1 0 1z 0z 0o 0z
1248.0 | 1 1 1 1 1 0 1z 0z 0z 0z
1300.0 | 0 1 1 1 1 0 1z 0o 0z 0z
```

```
QuickSim      User time scale =      1.0 nsec,  Input radix = Hex

FORCE BUSFREE 0
FORCE SCSI_ID4 1
FORCE SCSI_ID5 0
FORCE SCSI_ID6 0
FORCE SCSI_ID7 0
RUN 10
RUN 10
FORCE BUSFREE 1
RUN 80
RUN 800
RUN 400
#    Warning: Spike at time: 1230.0 on net '/BSY'
RUN 100
```

Figure 60 Timing simulation for routed ARBIT circuit with arbitration won

respectively. Figure 60 and 61 show the timing simulations of the routed ARBIT circuit for arbitration won and lost, respectively. An additional 49 ns is seen in the BSY and SEL



```
 845.0  1 1 1 0 1 0 1z 0z 0z 0z
 852.0  1 1 1 0 1 0 1z 0z 0z 0z
 870.0  1 1 1 0 1 0 1z 1  0z 0z
 900.0  0 1 1 0 1 0 1z 1  0z 0z
1000.0  1 1 1 0 1 0 1z 1  0z 0z
1100.0  0 1 1 0 1 0 1z 1  0z 0z
1200.0  0 1 1 0 1 0 1z 1  0z 0z
1250.0  1 1 0 0 1 0 1z 1  0z 0z
1300.0  0 1 0 0 1 0 1z 1  0z 0z
```

```
FORCE SCSI_ID4 1
FORCE SCSI_ID5 0
FORCE SCSI_ID6 0
FORCE SCSI_ID7 0
RUN 10
RUN 10
FORCE BUSFREE 1
RUN 50
RUN 800
FORCE DB5 1
RUN 400
    Warning: Spike at time: 1230.0 on net '/BSY'
RUN 100
```

Figure 61 Timing simulation for routed ARBIT circuit with arbitration lost

times due to routing delays. This is not a real problem since this just means that the SCSI will spend an extra 49 ns arbitrating the bus. Since the design arbitration time is 2000 ns, that extra time due to LCA implementation is insignificant. Figure 62 shows the schematic for the SELECT circuit which puts the logical OR of the SCSI ID and the TARGET ID on the SCSI bus. The selection phase begins when the SEL signal is asserted as a result of winning arbitration. Figure 63 shows the unrouted SELECT circuit timing simulation

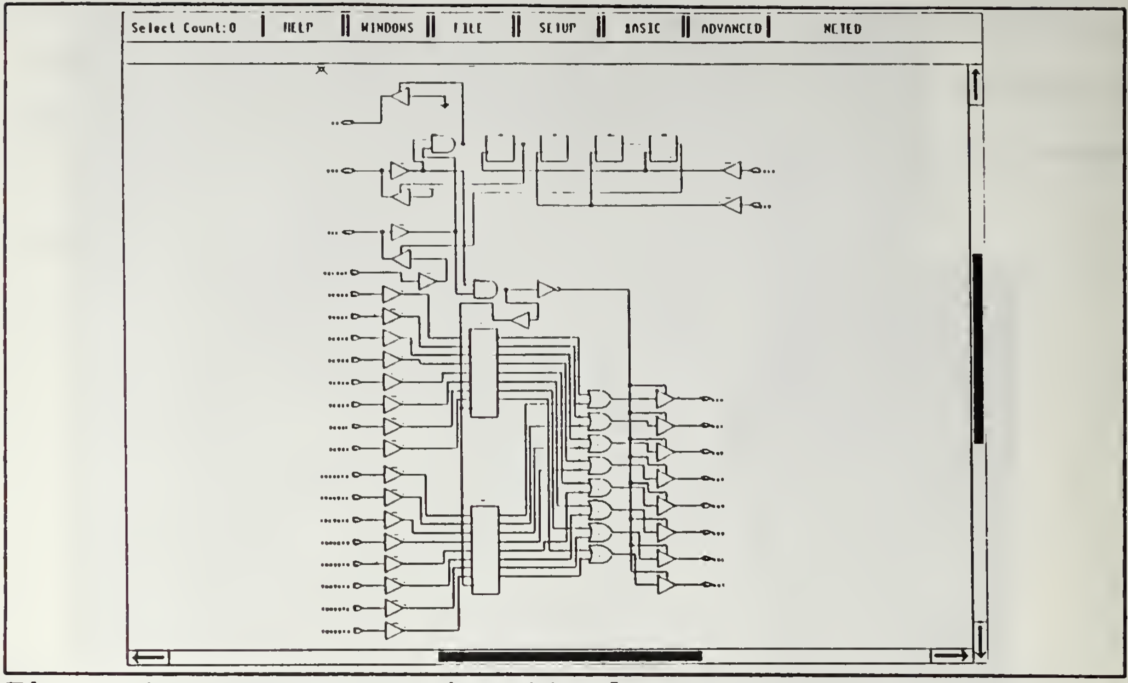
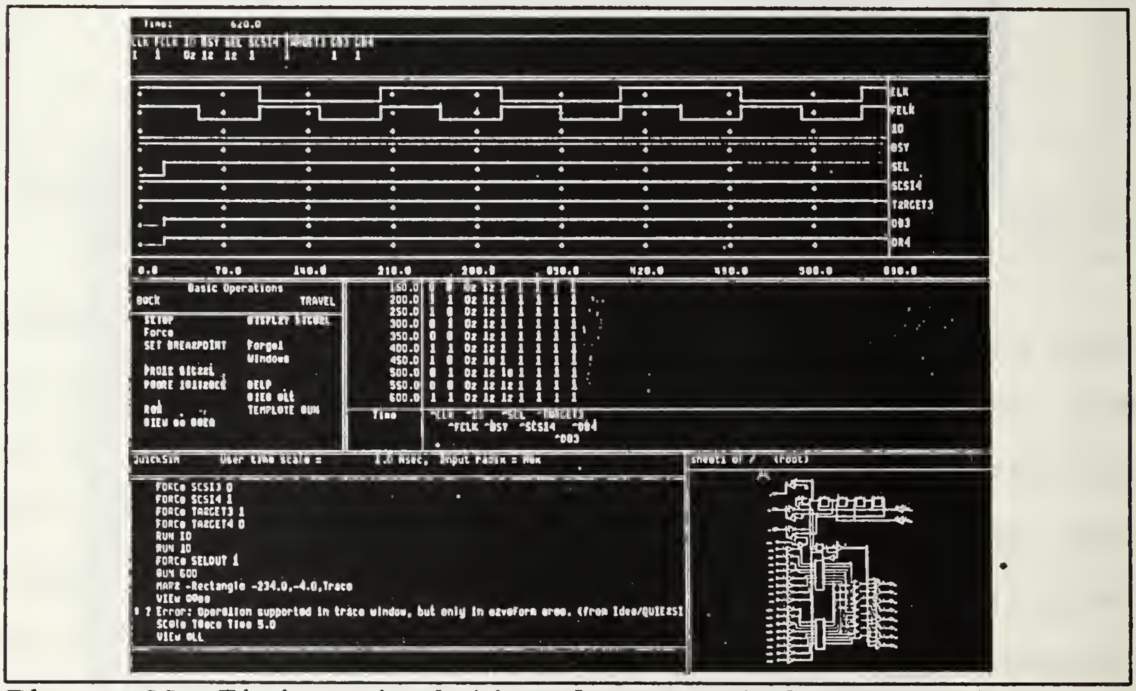


Figure 62 LCA_NETED schematic for SELECT circuit

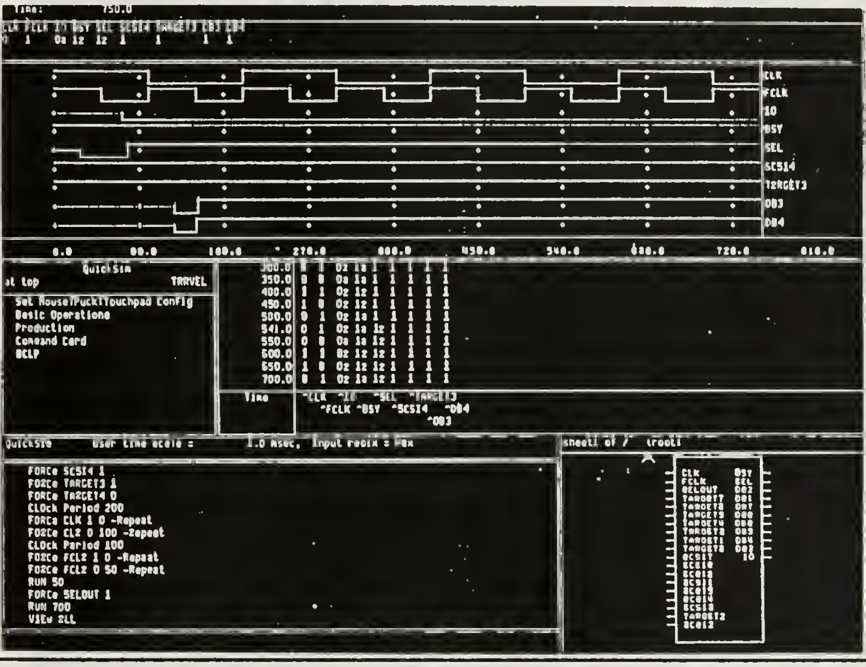


```
 150.0  0  0  0z 1z 1  1  1  1  1
 200.0  1  1  0z 1z 1  1  1  1  1
 250.0  1  0  0z 1z 1  1  1  1  1
 300.0  0  1  0z 1z 1  1  1  1  1
 350.0  0  0  0z 1z 1  1  1  1  1
 400.0  1  1  0z 1z 1  1  1  1  1
 450.0  1  0  0z 10 1  1  1  1  1
 500.0  0  1  0z 1z 10 1  1  1  1
 550.0  0  0  0z 1z 1z 1  1  1  1
 600.0  1  1  0z 1z 1z 1  1  1  1
 Time   ^CLK  ^IO   ^SEL  ^TARGET3
          ^FCLK ^BSY  ^SCSI4  ^OB4
                              ^OB3
```

```
QuickSim      User time scale =      1.0 Nsec,  Input radix = Hex
   FORCe SCSI3 0
   FORCe SCSI4 1
   FORCe TARGET3 1
   FORCe TARGET4 0
   RUN 10
   RUN 10
   FORCe SELOUT 1
   RUN 600
   MARK -Rectangle -234.0,-4.0,Trace
   VIEw [illegible]
# ? Error: Operation supported in trace window, but only in waveform area. (from Idea/QUIESI
   SCale TRace Time 5.0
   VIEw ALL
```

Figure 63 Timing simulation for unrouted SELECT circuit

which puts the required information on the SCSI bus without



```
  300.0  0  1  02 1a 1  1  1  1  1
  350.0  0  0  0a 1a 1  1  1  1  1
  400.0  1  1  02 12 1  1  1  1  1
  450.0  1  0  02 12 1  1  1  1  1
  500.0  0  1  02 1a 1  1  1  1  1
  541.0  0  1  02 1a 12 1  1  1  1
  550.0  0  0  0a 1a 12 1  1  1  1
  600.0  1  1  02 12 12 1  1  1  1
  650.0  1  0  02 12 12 1  1  1  1
  700.0  0  1  02 1a 12 1  1  1  1
   Time  ^CLK  ^IO  ^SEL  ^TARGET3
           ^FCLK ^BSY ^SCSI4  ^DB4
                            ^DB3
```

```
QuickSim    User time scale =    1.0 nsec,  Input radix = hex

    FORCe SCSI4 1
    FORCe TARGET3 1
    FORCe TARGET4 0
    CLOck Period 200
    FORCe CLK 1 0 -Repeat
    FORCe CLK 0 100 -Repeat
    CLOck Period 100
    FORCe FCLK 1 0 -Repeat
    FORCe FCLK 0 50 -Repeat
    RUN 50
    FORCe SELOUT 1
    RUN 700
    VIEw ALL
```

Figure 64 Timing simulation for routed SELECT circuit

delay. Figure 64 shows the routed SELECT circuit timing simulation which shows a 60 ns delay in getting the IDs on the SCSI bus.

Figure 65 shows the schematic for the SELTAR circuit which acknowledges the SCSI device selection as a target by asserting the BSY line. Figure 66 shows the unrouted SELTAR circuit timing simulation which shows the BSY line being asserted immediately upon selection whereas Figure 67 shows the routed SELTAR circuit timing simulation that shows BSY being asserted 67 nS after selection. This completes the verification of the BUSFREE, ARBITRATION and SELECTION phases of the SCSI operation.

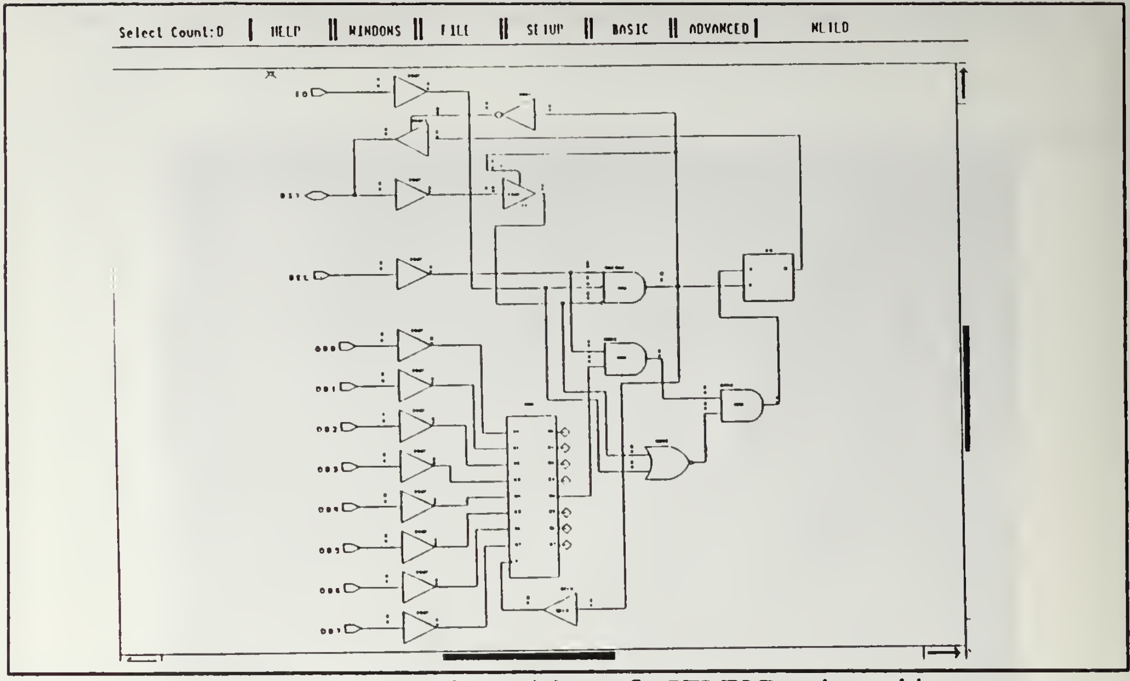
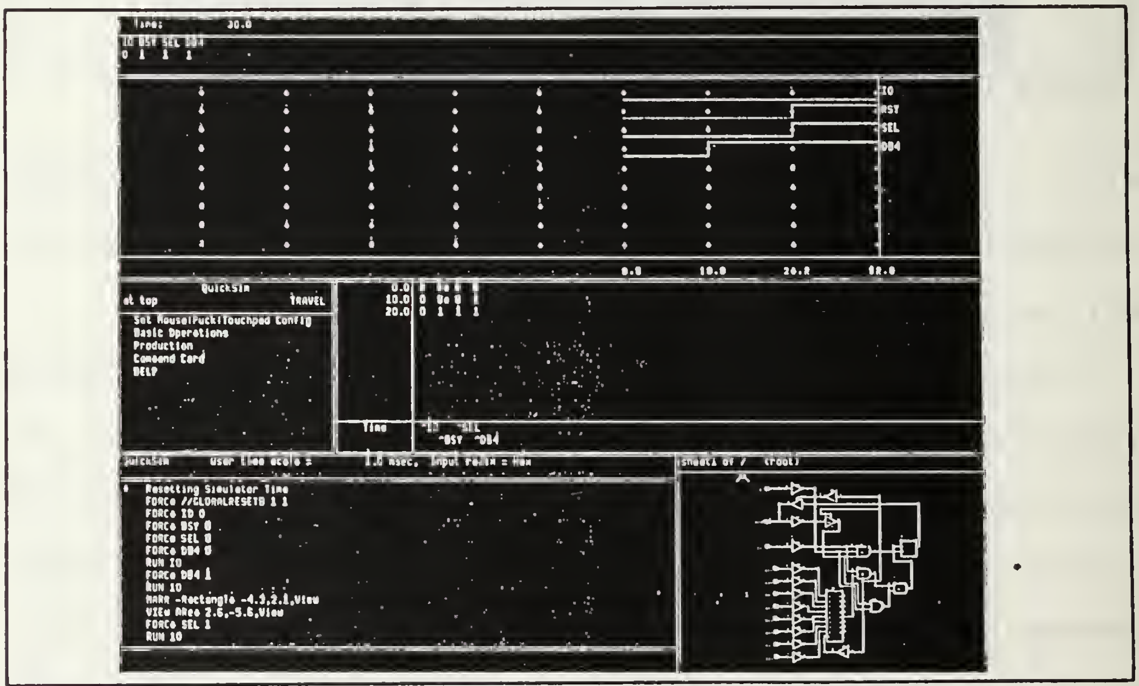
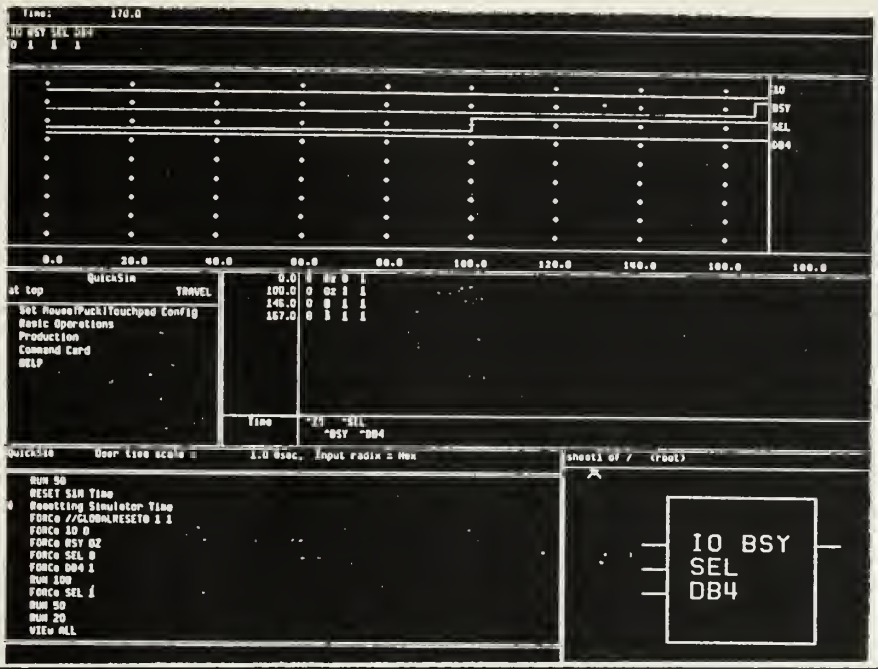


Figure 65 LCA_NETED schematic of SELTAR circuit



```
Time:          30.0
IO BSY SEL DB4
0  1   1   1

        0.0  10.0  20.0  30.0

 0.0 | 0  0o 0  0
10.0 | 0  0o 0  1
20.0 | 0  1  1  1
Time   IO  SEL
         BSY  DB4

QuickSim     User time scale =     1.0 nsec,  Input radix = Hex

*  Resetting Simulator Time
   FORCe //GLOBALRESETB 1 1
   FORCe IO 0
   FORCe BSY 0
   FORCe SEL 0
   FORCe DB4 0
   RUN 10
   FORCe DB4 1
   RUN 10
   MARk -Rectangle -4.3,2.1,View
   VIEw ARea 2.6,-5.6,View
   FORCe SEL 1
   RUN 10
```

Figure 66 Timing simulation of unrouted SELTAR circuit



```
Time:        170.0
IO BSY SEL DB4
0  1   1   1
```

```
  0.0 | 1 0Z 0 1
100.0 | 0 0Z 1 1
146.0 | 0 0  1 1
167.0 | 0 1  1 1
 Time | ^IO  ^SEL
        ^BSY ^DB4
```

```
QuickSim     User time scale =     1.0 nsec.  Input radix = Hex
   RUN 50
   RESET SIM Time
0  Resetting Simulator Time
   FORCe //GLOBALRESETO 1 1
   FORCe IO 0
   FORCe BSY 0Z
   FORCe SEL 0
   FORCe DB4 1
   RUN 100
   FORCe SEL 1
   RUN 50
   RUN 20
   VIEw ALL
```

Figure 67 Timing simulation of routed SELTAR circuit

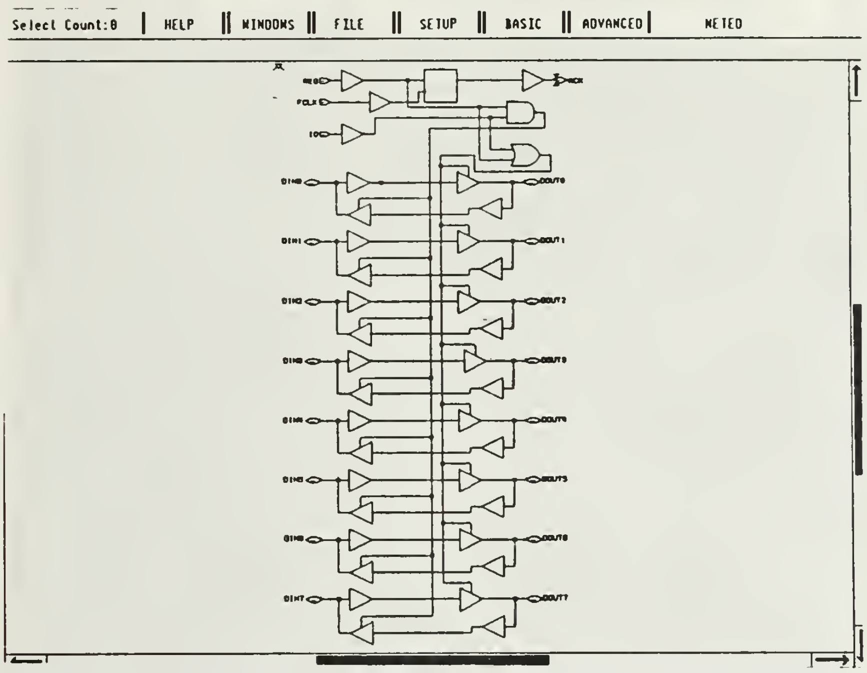
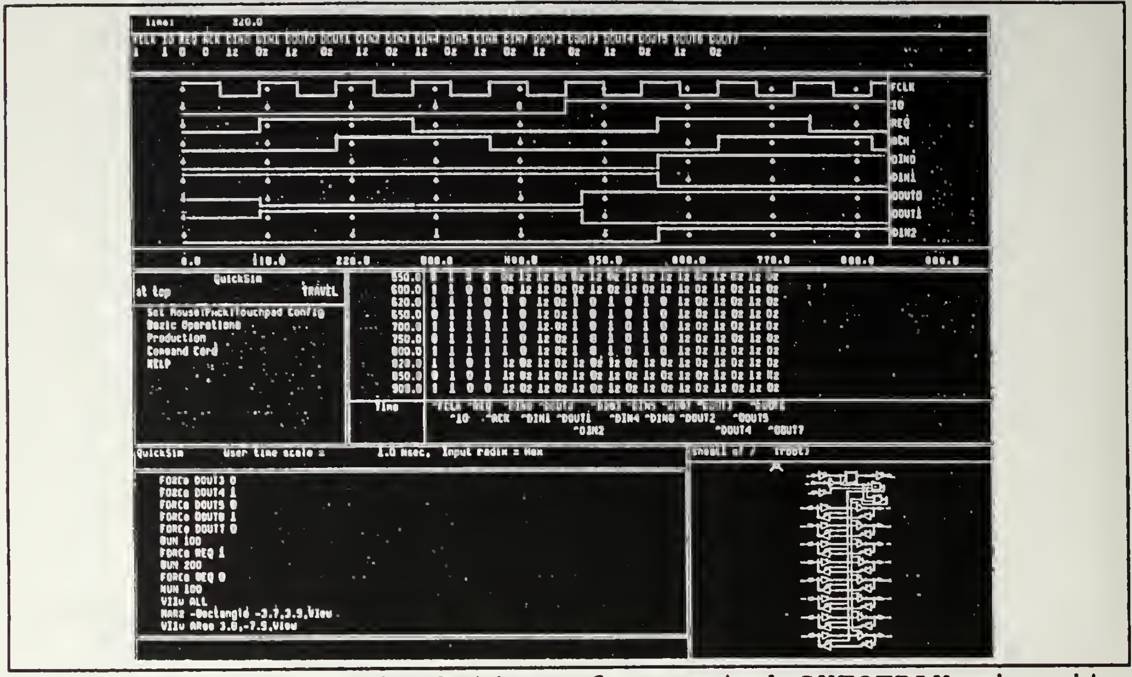


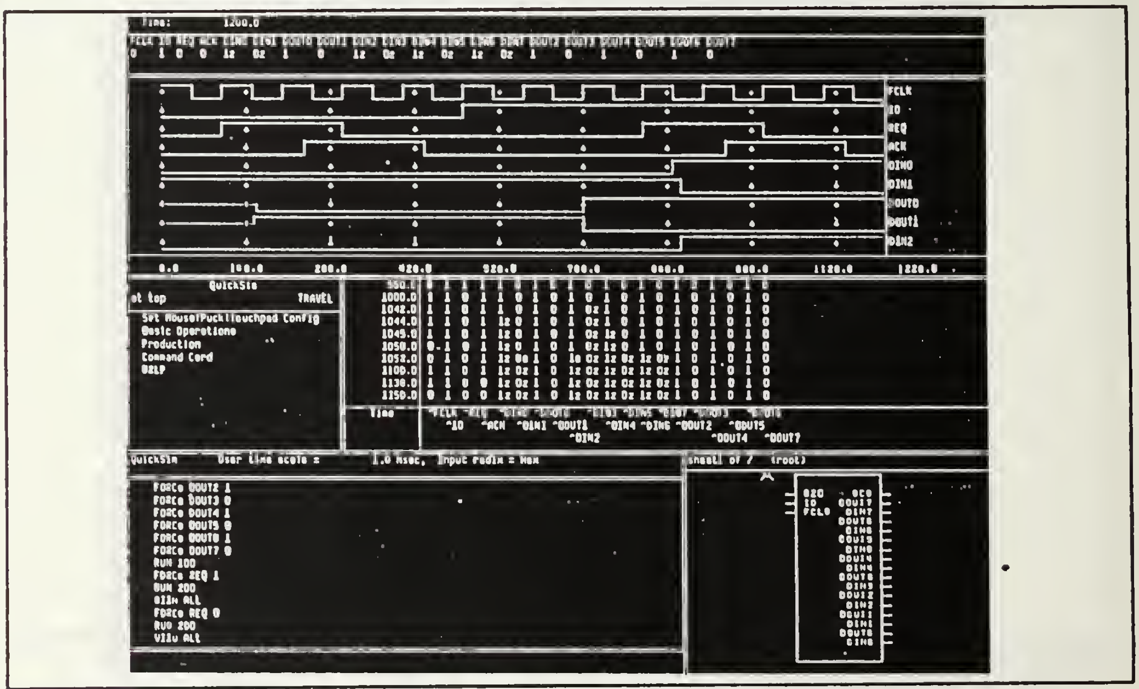
Figure 68 LCA_NETED schematic of INFOTRAN circuit



```
QuickSim      User time scale =      1.0 Nsec,  Input radix = Hex
```

```
FORCe DOUT3 0
FORCe DOUT4 1
FORCe DOUT5 0
FORCe DOUT6 1
FORCe DOUT7 0
RUN 100
FORCe REQ 1
RUN 200
FORCe REQ 0
RUN 100
VIEw ALL
MARk -Rectangle -3.7,3.9,View
VIEw ARea 3.8,-7.5,View
```

Figure 69 Timing simulation of unrouted INFOTRAN circuit



```
QuickSim      User time scale =      1.0 Nsec,  Input radix = Hex
```

```
FORCe DOUT2 1
FORCe DOUT3 0
FORCe DOUT4 1
FORCe DOUT5 0
FORCe DOUT6 1
FORCe DOUT7 0
RUN 100
FORCe REQ 1
RUN 200
VIEw ALL
FORCe REQ 0
RUN 200
VIEw ALL
```

Figure 70 Timing simulation of routed INFOTRAN circuit

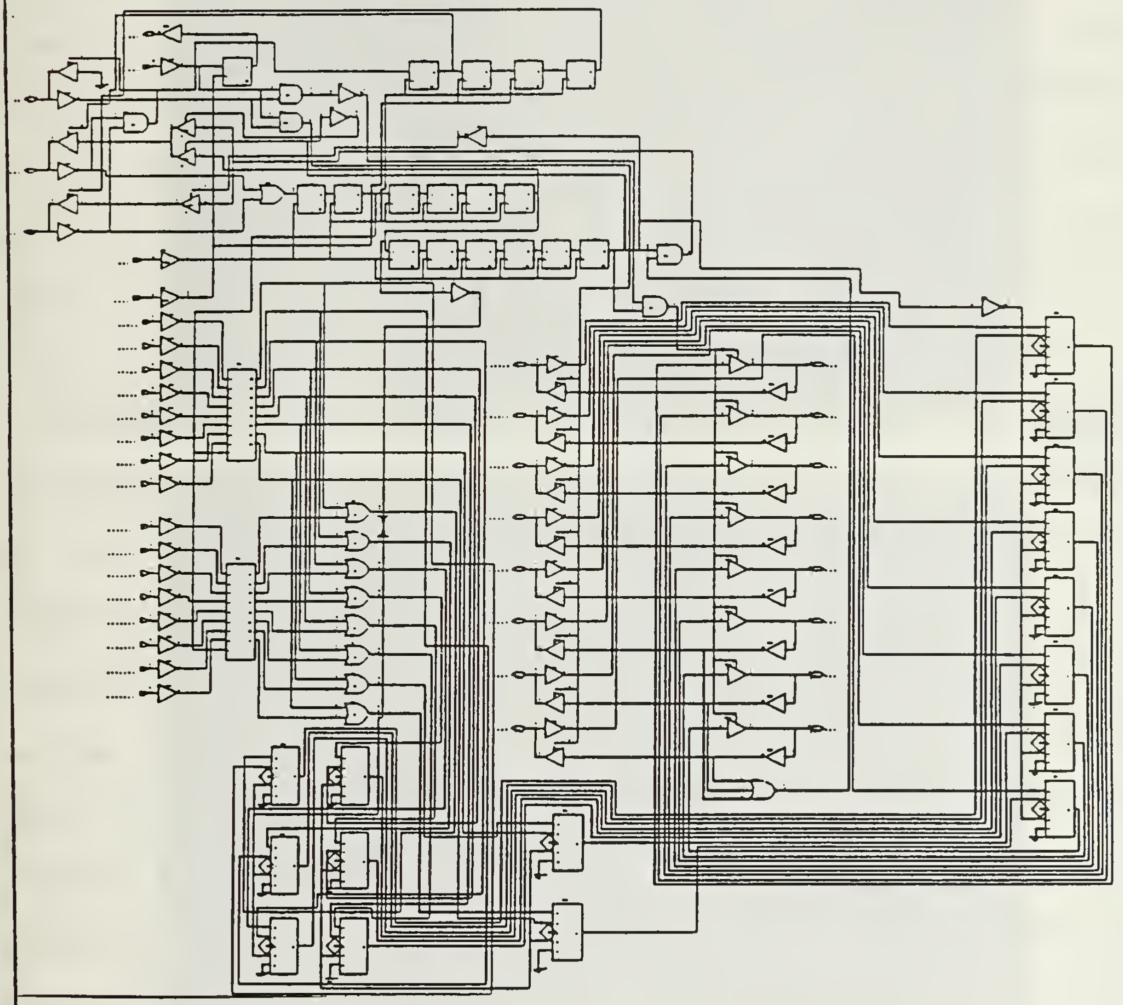
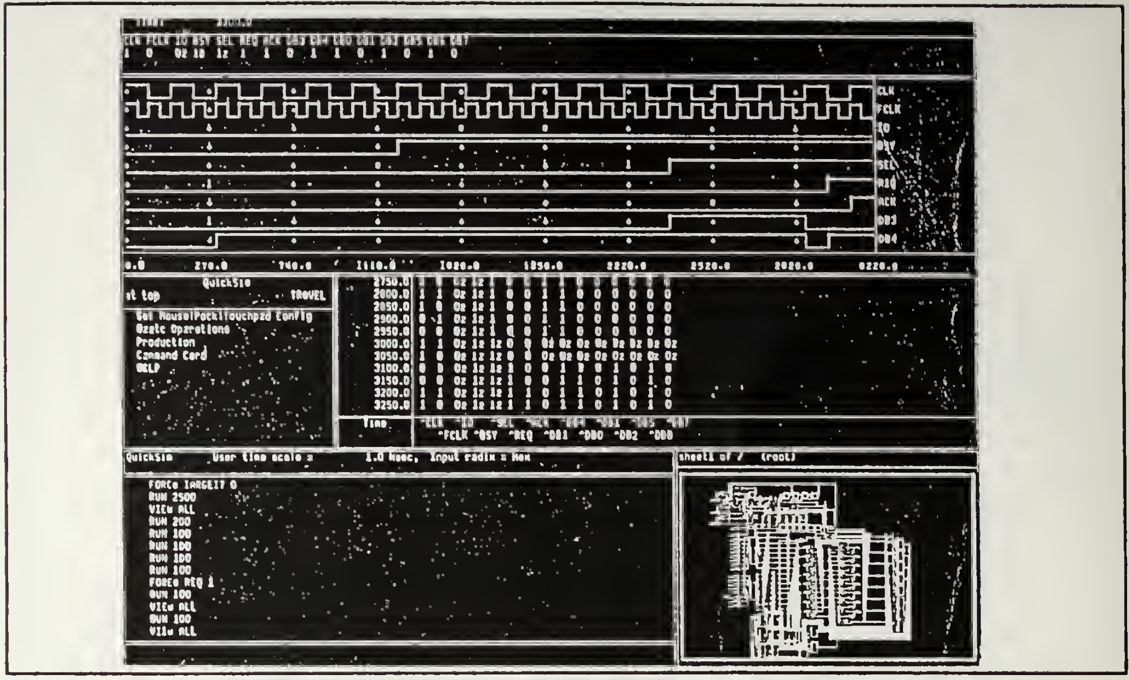
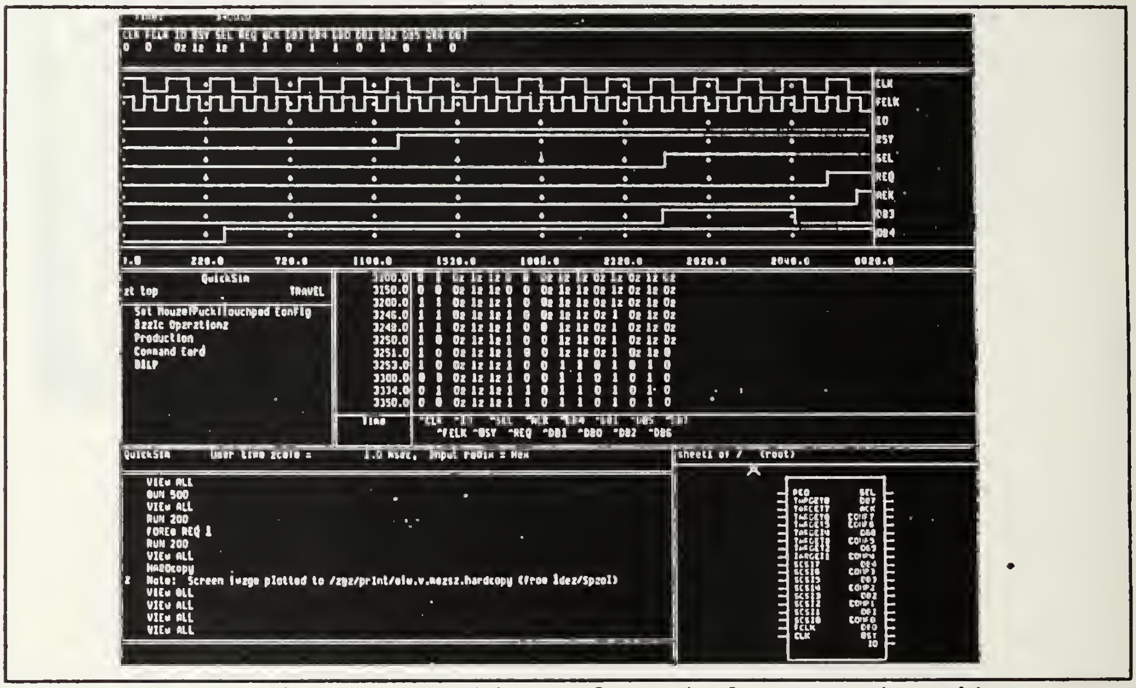


Figure 71 LCA_NETED schematic for completed SCSI design



```
QuickSim      User time scale =      1.0 Nsec,  Input radix = Hex

  2750.0
  2800.0  1 1 0z 1z 1  0  0  1  1  0  0  0  0  0  0
  2850.0  1 0 0z 1z 1  0  0  1  1  0  0  0  0  0  0
  2900.0  0 1 0z 1z 1  0  0  1  1  0  0  0  0  0  0
  2950.0  0 0 0z 1z 1  0  0  1  1  0  0  0  0  0  0
  3000.0  1 1 0z 1z 1z 0  0  0z 0z 0z 0z 0z 0z 0z 0z
  3050.0  1 0 0z 1z 1z 0  0  0z 0z 0z 0z 0z 0z 0z 0z
  3100.0  0 0 0z 1z 1z 1  0  0  1  1  0  1  0  1  0
  3150.0  0 0 0z 1z 1z 1  0  0  1  1  0  1  0  1  0
  3200.0  1 1 0z 1z 1z 1  1  0  1  1  0  1  0  1  0
  3250.0  1 0 0z 1z 1z 1  1  0  1  1  0  1  0  1  0
    Time
            ^FCLK ^BSY  ^REQ  ^DB1  ^DB0  ^DB2  ^DB0

    FORCe TARGET7 0
    RUN 2500
    VIEw ALL
    RUN 200
    RUN 100
    RUN 100
    RUN 100
    RUN 100
    FORCe REQ 1
    RUN 100
    VIEw ALL
    RUN 100
    VIEw ALL
```

Figure 72 Timing simulation of unrouted SCSI circuit



```
QuickSim      User time scale =      1.0 Nsec,  Input radix = Hex

  3100.0
  3150.0  0 0 0z 1z 1z 0  0  0z 1z 1z 0z 1z 0z 1z 0z
  3200.0  1 1 0z 1z 1z 1  0  0z 1z 1z 0z 1z 0z 1z 0z
  3246.0  1 1 0z 1z 1z 1  0  0z 1z 1z 0z 1  0z 1z 0z
  3248.0  1 1 0z 1z 1z 1  0  0  1z 1z 0z 1  0z 1z 0z
  3250.0  1 0 0z 1z 1z 1  0  0  1z 1z 0z 1  0z 1z 0z
  3251.0  1 0 0z 1z 1z 1  0  0  1z 1z 0z 1  0z 1z 0
  3253.0  1 0 0z 1z 1z 1  0  0  1  1  0  1  0  1  0
  3300.0  0 0 0z 1z 1z 1  0  0  1  1  0  1  0  1  0
  3334.0  0 1 0z 1z 1z 1  1  0  1  1  0  1  0  1  0
  3350.0  0 0 0z 1z 1z 1  1  0  1  1  0  1  0  1  0
    Time
            ^FCLK ^BSY  ^REQ  ^DB1  ^DB0  ^DB2  ^DB6

    VIEw ALL
    RUN 500
    VIEw ALL
    RUN 200
    FORCe REQ 1
    RUN 200
    VIEw ALL
    HARDcopy
z   Note:  Screen image plotted to /zgz/print/elw.v.mezsz.hardcopy (from 1dez/Spzol)
    VIEw ALL
    VIEw ALL
    VIEw ALL
    VIEw ALL
```

Figure 73 Timing simulation of routed SCSI circuit

Figure 68 shows the schematic of the INFOTRAN circuit. The direction of data transfer is a function of signal IO. When IO is low, data will be sent from initiator to target, and when IO is high, data transfer will go from target to initiator. The REQ signals data transfer to start and the ACK signal indicates the presence of valid data on the data bus. Figure 69 shows the timing simulation of the unrouted INFOTRAN circuit which shows bidirectional data transfer as well as the REQ/ACK handshake. A 100 ns REQ/ACK handshake delay was chosen to account for the cable skew and deskew delays (55 ns) plus the delay required by the SCSI standard as well as anticipated routing delays. The goal was not to acknowledge the REQ assertion until at least 55 ns after valid data is on the SCSI bus. Figure 70 shows the timing simulation of the INFOTRAN circuit which again shows a bidirectional data transfer and the ACK being asserted 60 ns after valid data is on the SCSI bus which is consistent with the 55 ns delay required by the SCSI standard. Upon completing thorough testing of all parts of the SCSI the parts were put into one schematic for implementation into an LCA. Figure 71 shows the completed design. Multiplexors were used to route different types of data to the external SCSI bus depending upon the state of the machine. Specifically, during the arbitration phase the SCSI ID goes to DB0-DB7 while in the selection phase the logical OR of the SCSI ID and TARGET ID is directed to the SCSI bus. During the information transfer phase data to (from) the initiator is sent to (received from) the SCSI bus. Figure 72 shows the timing simulation for the unrouted SCSI design. In this simulation, the bit pattern "00001000" was used for the SCSI ID while the TARGET ID bit pattern was "00010000". The data to be transfered by the SCSI during the information transfer phase was "10101010". These bit patterns were chosen in order to minimize the number of traces that had to be monitored in QUICKSIM to ensure proper operation of the SCSI. In this example, DB3 and DB4 were monitored. During the arbitration phase, the SCSI ID is placed on the SCSI bus which corresponds to DB3 and DB4 of 0 and 1, respectively. Once the selection phase starts, the SCSI bus carries the logical OR of the SCSI ID and the TARGET ID which in this case corresponds to DB3 and DB4 both equal to 1. The information transfer phase begins with the asserting of REQ by the target. In this example, DB3 and DB4 go to 0 and 1, respectively. Other events of significance are the assertion of BSY at 1200 ns and the assertion of SEL at 2400 ns. During the selection phase, IO is released at 2400 ns, BSY is released at 2600 ns, and SEL is released at 3000 ns. At this point information transfer can begin. Figure 73 shows the timing simulation for the routed and backannotated design. The same events occur in this simulation with the exception of a 49 ns delay. This implies a 1.6 % increase in time for this design to go through the bus free cycle, arbitrate the SCSI bus, and select the target. Once selection has occured, data is placed on the

SCSI bus upon assertion of REQ by the target. ACK is not asserted until 81 ns after all data on the SCSI bus is valid which exceeds the SCSI bus hold time requirements (45 ns). Specifically, the SCSI standard requires 55 ns for cable skew and deskew delays plus a bus hold time of 45 ns prior to asserting ACK to signal that there is valid data on the SCSI bus (total delay is 100 ns). This implies the fastest rate at which this design could transfer information is 5 MB/second. This actual transfer rate in this design is 3.73 MB/second as a result of the extra 34 ns delay between the REQ and ACK assertions. This extra 34 ns delay could be alleviated from this design by running FCLK at a higher clock rate (15.2 MHz vice 10 MHz). This performance could be improved by using a non-symetrical REQ/ACK handshake scheme. In this design a conservative approach was used to handle the handshake. If a non-symetrical handshake (i.e., when REQ is negated by the target ignore cable skew and deskew delays) the bandwidth could be doubled. This would involve some additional combinational logic circuitry to handle this method since the situation of transfer from target to initiator would be reversed. Making the REQ/ACK handshake symetrical transfers in both directions are handled by the same circuitry. The tradeoff was simplicity for bandwidth.

Once the SCSI design has been implemented into a placed and routed ".lca" file, it must be converted into a bit stream and then programmed into an EPROM. On the Apollo

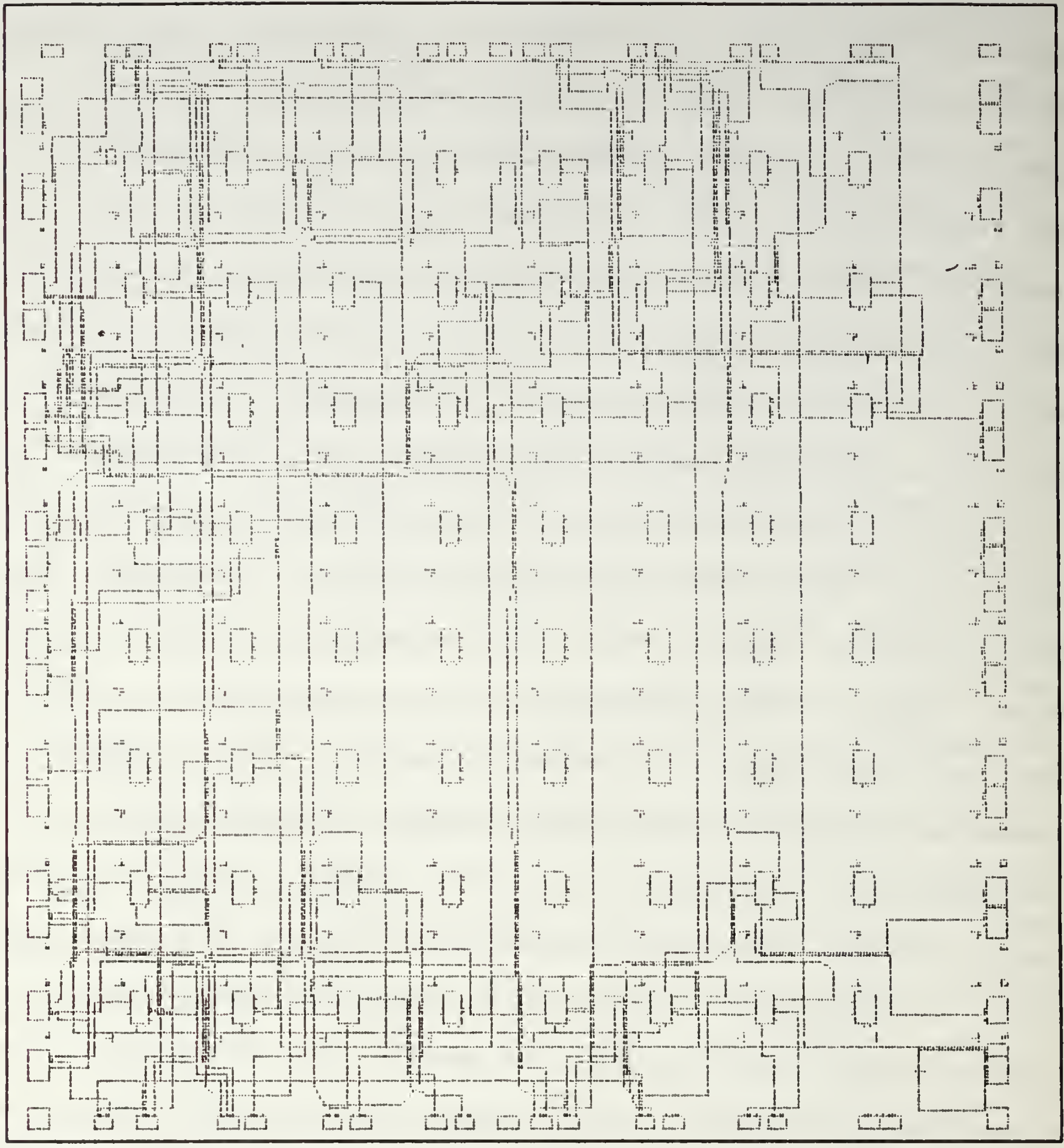


Figure 74 LCA placement and routing of SCSI design

workstation, this is accomplished by the MAKEBITS and MAKEPROM programs. These programs are invoked from the command line mode since there is no design manager. In the workstation environment there also is no capability to graphically use EDITLCA. To do this, the .lca file must be transfered into the PC environment. Figure 74 shows the file "scsi.lca" after being transfered to the PC which is then brought into the EDITLCA program. Once in EDITLCA, the design can be edited to change routing. This should not be done by an inexperienced designer or someone without a detailed knowledge of LCA architecture.

Once the SCSI design was successfully placed and routed with a satisfactory test cycle run, it was noted that the overall delay introduced by the technology was 49 ns. This corresponds to the SCSI cycle completing its bus free, arbitration and selection phases, and is prepared to transfer information. In testing the various parts of the SCSI, it was noted that the BUSFREE, ARBIT and SELECT circuits introduced a delay of 36 ns, 49 ns and 60 ns, respectively. This would lead to a total of 145 ns delay. This implies that the placement and routing algorithm is quite efficient in optimizing design overlap.

E. THE DESIGN VERIFICATION PROBLEM - PROTOTYPE VS BACKANNOTATED SIMULATION

Both prototype and backannotated simulation are valid methods of design verification. Both methods have their advantages and disadvantages. In this thesis, the backannotated simulation approach was stressed. However, before having a totally valid design, prototyping would need to be done. While backannotated simulation takes into account technology introduced anomolies, it is not a substitute for prototyping. One very attractive characteristic of the LCA technology is that prototyping can be done at a much later stage in the design process. This is because design flaws can be so easily corrected. In other technologies, some design prototyping would need to be done at an earlier stage in the design. For example, if standard TTL parts were being used in this design, before the entire design was combined, a prototype of the individual design parts would need to be completed. As part of this thesis, a prototyping board was constructed which configured an XC3020 LCA in Master Parallel Low mode. Once a design is implemented, the "design.rpt" file will tell the designer which LCA pins are being used for which signals. In this way, the designer will be able to actually test the combinational logic of the design. This prototyping board is however extremely limited in its resources. It requires a +5 VDC power supply. The LCA was found to function at a voltage as low as 2.9 VDC, but when trying to program via a download cable, the DONE signal was never recognized going high so the download programming cycle never completed. As a result, a +6 VDC power supply was used.

F. DESIGN CONSIDERATIONS

When utilizing this technology there is no substitute for experience. In this discussion, the assumption will be made that the design is being done in the engineering workstation environment. In starting any design, it is important to keep good documentation of what was done. This can be aided by having a good file directory for all design work. It is also important to keep track of what files have had what processes performed (i.e. which ".lca" files have been routed). In inputing designs into the XILINX development environment for LCA implementation, it is very improtant that these designs have been created with the XILINX macros and parts from the supplied libraries. While it is true that the XILINX macros utilize components from the Mentor Graphics gen_lib, they often attach properties which are necessary to process these designs into the XILINX system.

VI. CONCLUSIONS

From the work done in this thesis, several conclusions can be made. Implementation of designs into FPGAs is an efficient method of implementing design work in a manner that is easily modified either to correct design errors or to improve design performance. The performance of designs implemented with this technology was comparable, as shown in the SCSI design example, to other currently available technologies. The XILINX implementation tools, while flexible and able to accept a variety of design input methods, required the use of the XILINX supplied libraries to function properly. Direct use of previously tested designs developed under the Mentor Graphics development system was not possible without major rework. This is not a fault of the XILINX development system but rather a characteristic. Any previously developed designs that were created prior to the installation of the XILINX software must be redone utilizing the XILINX macro libraries. The XILINX macro libraries were extensive and fulfilled all the needs of the design work done in this thesis. Additionally, the capability exists to write additional macros as desired. The XILINX development environment greatly reduced design development time. Minor design changes and design debugging could be done in nearly real time with a minimal amount of effort.

Design implementation worked best in the workstation environment because of increased machine speed (a factor of five) and fewer memory constraints which will allow for even larger design work in the future. The PC design platform is still necessary however, to utilize its graphics capabilities for manually editing or viewing LCA layout since the XACT program does not currently exist in the workstation environment. The XILINX software was extremely stable but did have a relatively steep learning curve. This was contributed to by the fact that the documentation for the workstation environment was first written for the PC environment and partially updated.

The LCA technology studied in this thesis is an important technology which should be persued with further study and use. Speculation into the possible uses of this technology is as wide as one can imagine. One very general use of this technology would be in any application where space and/or weight is a consideration; the LCA has the ability to be reconfigured to perform more than one function. Another possible use is when security is an issue; on power-down all device characteristic information is lost making reverse engineering of design virtually impossible.

LIST OF REFERENCES

1. XILINX Inc., The Programmable Gate Array Data Book, XILINX Inc., San Jose, CA, 1989.

2. XILINX Inc., XILINX Presents the XC4000 Technical Seminar, XILINX Inc., San Jose, CA, 1990.

3. R. Freeman, "User-programmable gate arrays," IEEE Spectrum, pp. 32-35, December 1988.

4. Draft Proposed American National Standard for Information Systems - Small Computer Systems Interface (SCSI), X3T9.2 Revision 17B, December 16, 1985.

5. XILINX Inc., Logic Cell Array Development System Reference Manual Vol. II, XILINX Inc., 1990.

497-269

Thesis
M554 Messa
c.1 Design implementation into field programmable gate arrays.